1-1-1979

Programming The RSA Public-Key Crvotosystem And Evaluation Of A Proposed Cryptanalytic Attack.

Gary C. Fisher

Programming The RSA Public-Key Cryptosystem And Evaluation Of A Proposed Cryptanalytic Attack

by

Gary C. Fisher

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Industrial Engineering

Lehigh University

1979

CERTIFICATE OF APPROVAL

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

Nov. 7th 1979

Date

Professor in Charge

Chairman of Department

ACKNOWLEDGMENTS

The author wishes to express his sincere appreciation to his major advisor Professor John W. Adams for suggesting the topic, and to his minor advisor Dr. Ben L. Wechsler for his able assistance.

A special thank you belongs to the author's parents for providing him with the opportunity.

Miss. Joanne Drela and family were also very helpful during the long period of writing this thesis, and the author wishes them to be aware of his appreciation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This thesis deals with the creation of a set of FORTRAN programs for use with the DECsystem-20 computer for implementing the public-key cryptosystem invented by R. Rivest, A. Shamir, and L. Adleman (RSA) of M.I.T. In addition a FORTRAN program was written to subject the cryptosystem to a cryptanalytic attack proposed by T. Herlestam.

A brief summary of the field of cryptology is provided. Use of conventional cryptology in providing privacy and security in computer and communications systems is reviewed. Recent research indicates that the best commercially available cryptographic system, the National Bureau of Standards Data Encryption Standard, may not provide adequate protection for many more years due to the possibility of building an inexpensive machine

utilizing microprocessors capable of performing the cryptanalysis using exhaustive search. Discussed is the implications to key distribution protocols which recommend that a key be split into several pieces which are then distributed over several channels. It is concluded that if the chance of any portion of a DES key being compromised is significant then this is not a safe protocol considering the reduced keyspace over which an antagonist must perform a search.

Public-key cryptosystems are not plagued by the necessity of first distributing the cryptographic keys over a secure channel. This plus the fact that they provide a method of implementing signatures and receipts makes such a system ideal for use with an electronic mail system. The RSA public-key cryptosystem was found to be much slower than the DES. The RSA system requires the use of multiprecision integer arithmatic. The multiprecision routines used are presented. Key generation requires the ability to find prime numbers at least 40-digits long for adequate security. A program was written that can find prime numbers of this magnitude in less than half a minute of CPU time with high reliability. The process for encrypting a plaintext message M into a ciphertext C is:

$$C \equiv M^e \pmod{n}$$

the decryption function is:

$$M \equiv C^d \pmod{n}$$

where n is the product of two large primes; e and d chosen by special criteria. Based on experimental results it was found that e and d should have a high proportion of zeroes in their binary representations for rapid encryption/decryption. Encryption/decryption time was found to increase linearly with the size of n and linearly, but discontinuously with the size of e and d.

The system developed is not useful for a high traffic volume system. It could be used to inexpensively secure an insecure channel for the distribution of the keys of a conventional cryptosystem. A suggestion is made for using a public-key cryptosystem to provide signature capability while security and privacy are provided using a conventional cryptosystem.

The experiments using Herlestam's cryptanalytic attack were not successful at breaking the RSA cryptosystem. Only in trivial cases where very unrealistically small n were used was there the slightest success. It is concluded that for large n a cryptanalytic attack based upon factoring n would be less costly, and more likely to succeed.

CHAPTER 1

AN OVERVIEW OF CRYPTOLOGY

1.1 INTRODUCTION

Cryptology, despite its ever increasing usefulness as a method of insuring the security of information communicated over many kinds of systems, e.g., military, diplomatic, and business, remains an arcane branch of human knowledge. The following overview of cryptology is presented so that the reader, who is unfamiliar with this field, may better follow the discussion in later chapters.

Cryptology encompasses two main branches. These are, in the respective order in which they logically would have developed, CRYPTOGRAPHY and CRYPTANALYSIS . Cryptography (from the Greek kryptos, "secret", and graphein, "to write") may be defined as the science or art of conveying a message, known as PLAINTEXT, in a form, known as a

CRYPTOGRAM, which is unintelligible to anyone not possessing knowledge of how the transformation of the message was accomplished. The practitioner of cryptography is called a CRYPTOGRAPHER.

Cryptanalysis may be defined as the science or art of extracting the meaning from a cryptogram without the knowledge of how the cryptogram was constructed, and/or discerning the actual technique (algorithm and key) used to convert a given plaintext into its corresponding cryptogram. The practitioner is called a CRYPTANALYST.

Cryptology is an old field that has probably been practiced in one form or another since man first began communicating his thoughts in writing. References to cryptographic procedures, and examples of cryptography have been found in the Bible (in Jeremiah XXV.-26, LI.-41, and in Isaiah VII.-6 the cryptographic name of the city of Babel is Sheshach) see Milliken [1], in Plutarch, and in the fifth century B.C. records of the Persian court [2]. For a fascinating and thorough history of cryptology and cryptanalysis the reader is directed to a book by David Kahn [3].

The main task of cryptography is to transform a plaintext message into a cryptogram, also known as a CIPHERTEXT when the cryptographic transformation used is a cipher, by a cryptographically secure method. That is, the method must withstand intense efforts at cryptanalysis. The more secure, the higher the WORK FACTOR' is for the cryptanalyst. Plaintext may be converted to a cryptogram by either of two major classes of CRYPTOSYSTEM. A message may be ENCIPHERED (ENCRYPTED) using a CIPHERSYSTEM, or it may be ENCODED using a CODE SYSTEM.

1.2 CODE SYSTEMS

In a code system the components of the plaintext, e.g., words, phrases, numbers are related to their equivalent code group by means of a code book or dictionary. No fixed relationship is maintained between the number of symbols in the plaintext and in the encoded version known as CODETEXT.

Table 1.1 contains a portion of a code system based on an example given in [4]. Note the use of two common methods of frustrating the efforts of the cryptanalyst. One is the use of more than one codetext equivalent for a

ENCODING

PLAINTEXT	NULLS	CODETEXT
.	.	.
.	.	.
.	.	.
CRYPTANALYSIS	.	013
CRYPTANALYST	419	712
CRYPTOGRAM	802	984
CRYPTOGRAPHER	014	444
CRYPTOLOGY	882	002
.		.
.		.
.		.

DECODING

CODETEXT	PLAINTEXT
.	.
.	.
.	.
012	BOMBER
013	CRYPTANALYSIS
014	(NULL)
015	JAMMING
016	(NULL)
.	.
.	.
.	.

Table 1.1

A simple example of a 2 part code.

component of plaintext. These are known as HOMOPHONES. The second device is the use of codetext with no plaintext equivalent. These are known as NULLS.

Code systems have been used for reasons other than to provide security. The MORSE code was developed primarily to overcome the technical limitations of wire communications in the nineteenth century. The very earliest of commercial codes was developed by Lloyd's, the insurance company, in the late eighteenth century. It consisted of using a set of light signals for ship to ship communication [5]. Such telegraphic codes were not new, only their use for commercial applications. The versatility and value of code systems are limited because only plaintext messages for which there exist a codetext equivalent may be encoded.

1.3 CIPHER SYSTEMS

Cipher systems are superior to code systems for most applications. Cipher systems have two main components: an algorithm, and a KEY. The algorithm should define a very large set of mathematical procedures for transforming plaintext into ciphertext. The key is what specifies which of the numerous possible transformations is to be

used. In general a cryptographic key is a sequence of numbers or characters which is known only by the legitimate users of the cryptosystem. Idealy the key should be short for easy memorization or storage.

The algorithm may also be kept secret, but if this is a requirement, it is a sign of severe weakness in the cryptosystem. To be useful the algorithm should permit rapid transformation of plaintext to ciphertext. For each encrypting operation there must exist an inverse transformation, preferably rapid, that changes the ciphertext back to plaintext. This operation is known as DECIPHERMENT or DECRYPTMENT.

Equation 1.1 is a symbolic mathematical representation of a cryptosystem as a single parameter family of invertible transformations from a space of

$$F_k\{M\} \rightarrow \{C\} \qquad (1.1)$$

plaintext messages {M} to a space of ciphertext messages {C}. The parameter is k, the key, and is selected from the set {K} called the keyspace.

In general there are three main classes of ciphers: SUBSTITUTION CIPHERS, TRANSPOSITION CIPHERS, and a combination of these known as a PRODUCT CIPHER. These three classes cover all conventional cipher systems. During the 1970s a completely new class of cipher was discovered. This was the PUBLIC-KEY CIPHER and will be treated in a seperate chapter.

In a substitution cipher different characters are substituted for the plaintext characters. The characters used need not come from a different set from that which is used to construct the plaintext. As an example of a substitution cipher we could use a SHIFT or as it is sometimes called, Caesar cipher (after Julius Caesar who is said to have used it). In such a cipher system the alphabet is offset to itself; the key being the amount of offset. Julius Caesar is said to have used a shift of three. Thus, emulating Julius Caesar, we get the substitution cipher system shown in Table 1.2.

```
PLAINTEXT : ABCDEFGHIJKLMNOPQRSTUVWXYZ
CIPHERTEXT: DEFGHIJKLMNOPQRSTUVWXYZABC
```

Table 1.2

An example of a shift cipher.

Using such a cipher, the plaintext, GARY FISHER, becomes KDVB IMXLHV.

Such a cipher is very insecure and can easily be broken using a table of letter frequencies. Greater, but not much greater, security can be obtained by more randomly associating the letters of the plaintext alphabet with the letters of the ciphertext alphabet. One method of achieving this is to use what is known as a KEYWORD or KEYPHRASE. This is illustrated in Table 1.3 using KEYWORD as the keyword. A code is not a substitution cipher at a

PLAINTEXT : ABCDEFGHIJKLMNOPQRSTUVWXYZ
CIPHERTEXT: KEYWORDABCFGHIJLMNPQSTUVXZ

Table 1.3

An example of a shift cipher using a keyword.

grosser level of language, i.e., words or phrases rather than individual letters of the alphabet, because, unlike a code, a substitution cipher maintains a fixed relationship between the number of symbols in the plaintext and the ciphertext.

In transposition cipher systems the original characters are retained but their order of appearance is jumbled. One simple cipher of this type is to write the plaintext in blocks of some specified length (the key), and encrypt the plaintext by then rewriting the message in column order. For example, the plaintext GARY FISHER, using a key of two would be enciphered by first writing it as:

GA
RY
FI
SH
ER

and then rewriting it as GRFSHAYIHR.

Product ciphers were greatly advanced during World War II when very secure ciphers were developed using alternate steps or ROUNDS of transposition and substituion. Today product ciphers constitute the major area of research in conventional cryptography.

There is one other major method of classifying ciphers. Any cipher is either a STREAM CIPHER or a BLOCK CIPHER. Stream ciphers process the plaintext in small pieces, usually characters or, in the case of digital equipment, bits. The "one time pad" to be discussed shortly is a stream cipher. Block ciphers operate on blocks of text several characters long. A good block

cipher will encipher two very similar but different texts into two very different ciphertexts. In other words, the cryptosystem produces a strong intersymbol dependence between each symbol in a block of plaintext. Advanced block ciphers extend this intersymbol dependence between blocks. This is a technique known as BLOCK CHAINING. Block chaining is very useful for highly redundant and/or structured data.

The security of a cryptosystem is rarely provable by strict mathematical argument. During the sixteenth and seventeenth centuries many mathematical proofs were developed as new cryptosystems came along. Unfortunately such proved secure systems were constantly being broken. Mathematical proofs fell into disfavor; the only accepted proof of a cryptosystem's security being evidence of it having resisted intense cryptanalytic attack over many years [6]. Unfortunately few ciphers suitable for data encryption, and electronic mail systems are available which have been subjected to years of cryptanalytic attack.

1.4 CRYPTANALYSIS

A system is considered COMPUTATIONALLY SECURE if the cost of cryptanalysis is great enough to prevent an adversary from carrying out the computation necessary to break the cipher, i.e., there is a high work factor. If a system can withstand any amount of computation it is considered UNCONDITIONALLY SECURE.

Only one cipher system, the Vernam cipher or "one time pad", is known to be proved secure. Shannon accomplished this in a pioneering paper on information theory [7]. The Vernam cipher was patented in 1918, yet today it is the cryptosystem used on the Washington to Moscow hot line because no better system has been developed. The following example in modulo 26 arithmatic (any base may be used) will serve to illustrate the operation of the Vernam cipher. Once again using GARY FISHER as the plaintext. The first step is to assign a unique numerical value to each letter of the alphabet. If A=0, B=1, C=2,.....,Z=25 then the plaintext, GARY FISHER, becomes 6,0,12,24,5,8,13,7,4,12 ignoring blanks. Next a random key is generated consisting of numbers between 0 and 26 inclusive. As an example, assume the following ten numbers are produced from a random number generator:

5,17,22,1,23,6,16,5,14,24. In order to encipher the plaintext, take the numerical form of each of the plaintext letters and add to them the corresponding random number from the key. This sum is then divided by 26 and the remainder is the ciphertext. For deciphering the ciphertext number has its corresponding key number subtracted from it. The result is then divided by 26 and the remainder is the numerical value of the plaintext character. This entire operation is illustrated in Table 1.4. The security of this cipher resides in the complete randomness of the key which causes a complete lack of structure in the ciphertext. It should be apparent that pseudorandom number generators may safely be used to generate keys only if they pass the statistical tests for randomness, e.g., poker test, runs test, etc. with flying colors, in order to frustrate the cryptanalyst, and are known only by the legitimate users of the system. A pseudorandom number generator and its seed are identical to the key and must be afforded equal protection. A major problem with the Vernam cipher is that the key must be as long as the message.

In general there are three modes of cryptanalytic attack. A CIPHERTEXT ONLY attack is one in which the cryptanalyst only has access to actual pieces of

PLAINTEXT:	G	A	R	Y	F	I	S	H	E	R
NUMERICAL FORM:	6	0	12	24	5	8	13	7	4	12

ENCIPHER

KEY:	5	17	22	1	23	6	16	5	14	24
plus										
PLAINTEXT:	6	0	12	24	5	8	13	7	4	12
modulo 26=	—	—	—	—	—	—	—	—	—	—
CIPHERTEXT:	11	17	8	25	2	14	3	12	18	10

DECIPHER

CIPHERTEXT:	11	17	8	25	2	14	3	12	18	10
minus										
KEY:	5	17	22	1	23	6	16	5	14	24
modulo 26=	—	—	—	—	—	—	—	—	—	—
PLAINTEXT:	6	0	12	24	5	8	13	7	4	12

Table 1.4

An example of the Vernam cipher or "one time pad".

ciphertext. This represents the most difficult form of cryptanalysis since only through study of ciphertext must the key and, possibly the algorithm, be deduced. A KNOWN PLAINTEXT attack is one in which the cryptanalyst has access to pieces of ciphertext and the corresponding plaintext. A CHOSEN PLAINTEXT attack is one in which the cryptanalyst can submit plaintext of his own choosing to the cryptosystem and can then study the resulting ciphertext.

Of the three modes of attack the first is by far the most likely to occur. Substitution ciphers easily succumb to attacks of this kind. A large body of data has been collected over the years about such things as the frequency of occurance of each letter of the alphabet, each digraph, each trigraph, words of one letter, words of two letters, etc. in English and many other languages. See Millikin [1] for tables of such data.

If a cryptosystem is considered insecure against a known plaintext attack the system users are forced never to make plaintexts public and must, in fact, rephrase every message before public release. From a business standpoint such a system is not very practical since many messages which are initially secret may be slated for

eventual public release, e.g., new product announcements. The chosen plaintext attack is not easily implemented since it requires the cryptanalyst to first plant a plaintext in the system and then intercept the resulting cryptogram.

The last thirty years have seen both cryptography and cryptanalysis advance phenomenally primarily due to the advent of the electronic computer. The cryptographer has been in a constantly accelerating race with technology. The ever decreasing ratio of cost to computational power aids both the cryptologist and the cryptanalyst. The ability to break a cryptosystem by enumeration of all possible solutions has become a real threat due to the computer.

CHAPTER 2

COMPUTERS AND CRYPTOLOGY

2.1 SECURITY AND PRIVACY

Two important issues for any organization using a computer are security and privacy. The latter can only be insured when the former is well provided. The aim of security is to prevent something from being lost (stolen) or misused. For a computer system to be secure it is necessary a) to protect the system from environmental hazards, b) to protect the system from access by unauthorized users, c) to protect the system's stored data/information, and d) to protect the data/information while being transmitted over the system's communication system. Of the four items, (a-d), only item (a) can not benefit from cryptology.

2.2 USER ACCESS

There are two types of user access to a computer system. First there is access to the actual equipment (hardware), secondly there is access to the operating system. The difference is essentially the difference between getting into the machine room, and getting "on" the system to run programs. The issueing of passwords is cryptology used in its simplist form as an aid to security. Passwords can be viewed as simple codes. The password is merely the codetext equivalent to some plaintext message such as "I am a legitimate user". Passwords are used either to limit access to the equipment or to the operating system.

Intense physical security systems are often not practical nor desireable. This is particularly true of distributed processing systems were it would be extremly expensive to secure each remote terminal and each communications line by means of physical barriers; and where the legitimate users should have quick and easy access to the system.

Passwords are the most common method of preventing an unauthorized user from accessing a computer system. Wilkes [8] states that to be used, passwords must be

carefully guarded and changed frequently, the printing of passwords should be suppressed whenever a user logs onto a system; and he identifies the most vulnerable point of a password system as the list of passwords stored in the computer. In chapter 4 recently proposed cryptographic methods of securing the password table, based upon "one-way" functions, will be reviewed.

2.3 STORED DATA/INFORMATION

Government, military, business organizations, and private citizens are or should be concerned about the security of data/information contained in computer data bases and data banks. Good physical security systems such as a vault for tapes and/or disks is the first step in insureing that unauthorized access to data/information is prevented. Data encryption is gaining acceptance as a means of securing data/information recorded on magnetic storage media. The basic idea is to encrypt information before it is recorded and to decrypt it before use in some application. The security of the data depends upon the security of the cryptosystem and its keys. The advantage is that it is easier and cheaper to secure a key of a few bits than a data base of several million or more bits.

Ehrsam et al [9] distinguishes two classes of key one would encounter in a data encryption system. KEY-ENCRYPTING keys are used to encipher other keys and are defined during the process of initializing the cryptosystem. They are changed only at the end of long periods (about a year). DATA-ENCRYPTION keys are generated dynamically and exist only as long as the data they protect. For file security it is possible that the data-encryption key exist for a relatively long time. Any data encryption system may use several subclasses of each of these two main classes of key.

2.4 DATA TRANSMISSION

Communication security has benifited most from cryptology and is expected to play an ever increasing role as commercial applications, e.g., electronic funds transfer (EFT) develope. Cryptology can provide a means of securing a communications channel so as to provide two things to the users: privacy, and the capability of authenticating messages. When information can not be extracted by unauthorized parties from a message sent over a channel connecting two or more nodes of a system, the channel is said to provide privacy. If unauthorized

injection of messages is not possible the receiver of a message is assured of the legitimacy of the sender; and the channel is said to provide authentication capability.

Ehrsam et al [9] define three possible methods of incorporating encryption into a communications system: LINK-BY-LINK, NODE-BY-NODE and END-TO-END encryption.

With link-by-link encryption Figure 2.1 the data moving between two system nodes is encrypted across the channel connecting the nodes. It is logically independent of the computer system, and does not necessarily imply the existence of a cryptographic capability at the communicating nodes. It can be thought of as implemented by a pair of devices, one at each end of the channel, located between the two communicating nodes and their modems (modulators or demodulators) for accomplishing the cryptographic transformation. There is a unique key to protect the data moving between each pair of nodes, therefore messages that traverse several nodes must undergo a series of translations from one key to another.

Node-by-node encryption Figure 2.2 also protects each communications channel between nodes by a unique key. However the translation from one key to another occurs inside a security device module, which may be peripherally

attached to the node. Plaintext exists only within the security device, not within the node.

With end-to-end encryption Figure 2.3 the data is encrypted at the originating node and is not decrypted until it arrives at its final destination. This should be contrasted with node-by-node where a message will occur in clear form at each node, but only within the security device, and with link-by-link where every channel that carries encrypted messages must be bracketed by standalone cryptographic devices. The cryptographic capability is integrated into the participating nodes in a way that allows the system to control the setting of keys and turning the cryptosystem on and off. If the encryption capability is provided by the host CPU then the capability exists to encrypt system resident data, not just data in transit as is the case with the other two methods. Since with end-to-end encryption the routing information (provided in a header to the message signal) must remain in clear form, it is susceptible to TRAFFIC ANALYSIS - an intelligence gathering technique which attempts to extract useful information from the quantity and direction of messages in a system.

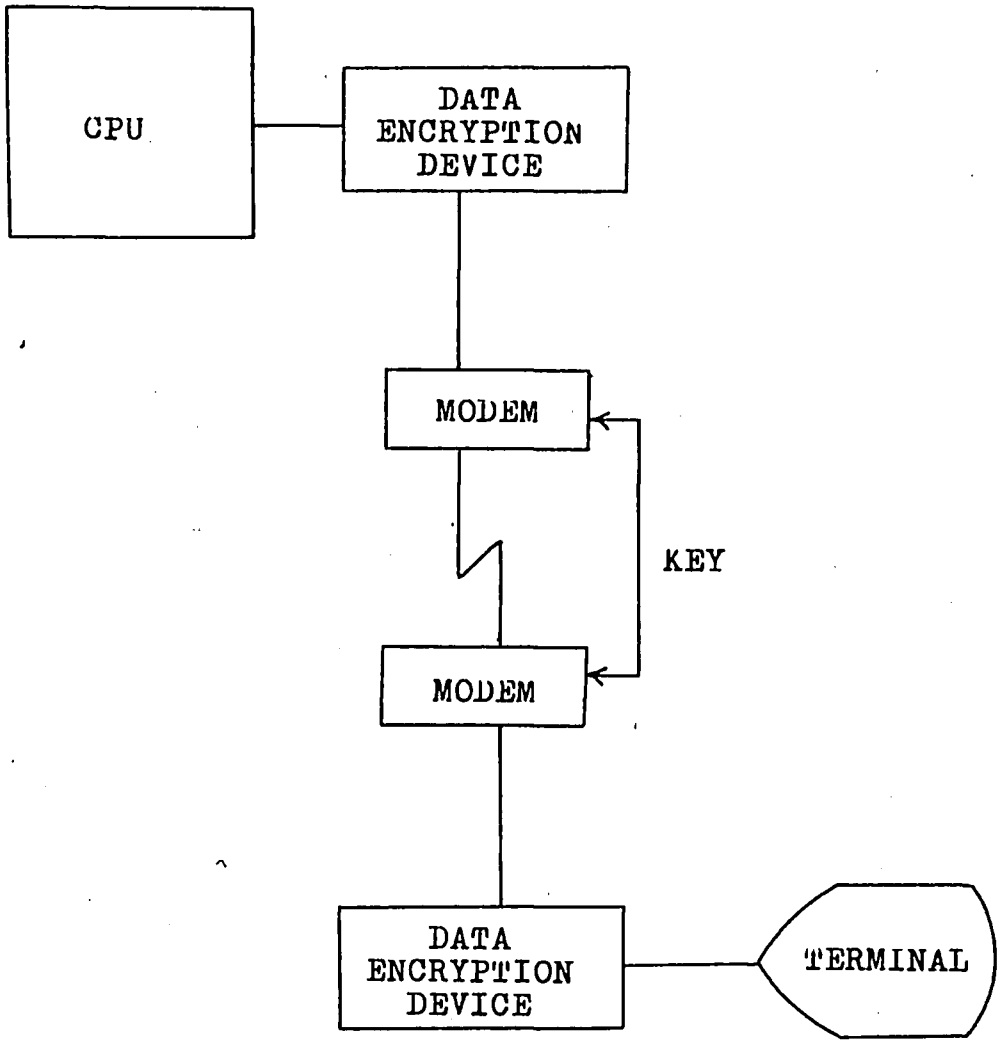


Figure 2.1

LINK-BY-LINK ENCRYPTION

Data is encrypted across the medium connecting two directly communicating nodes (CPU and TERMINAL).

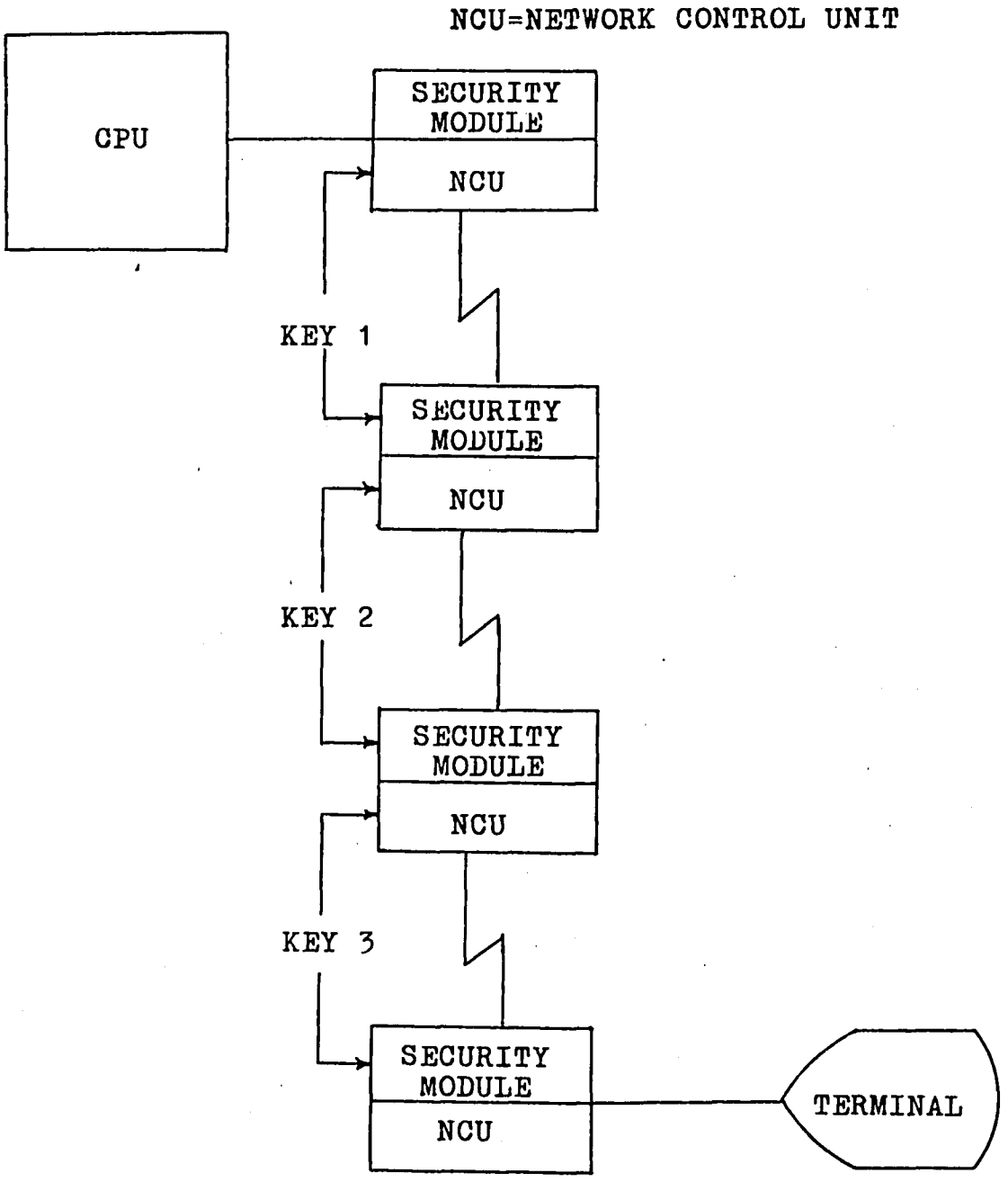


Figure 2.2

NODE-BY-NODE
ENCRYPTION

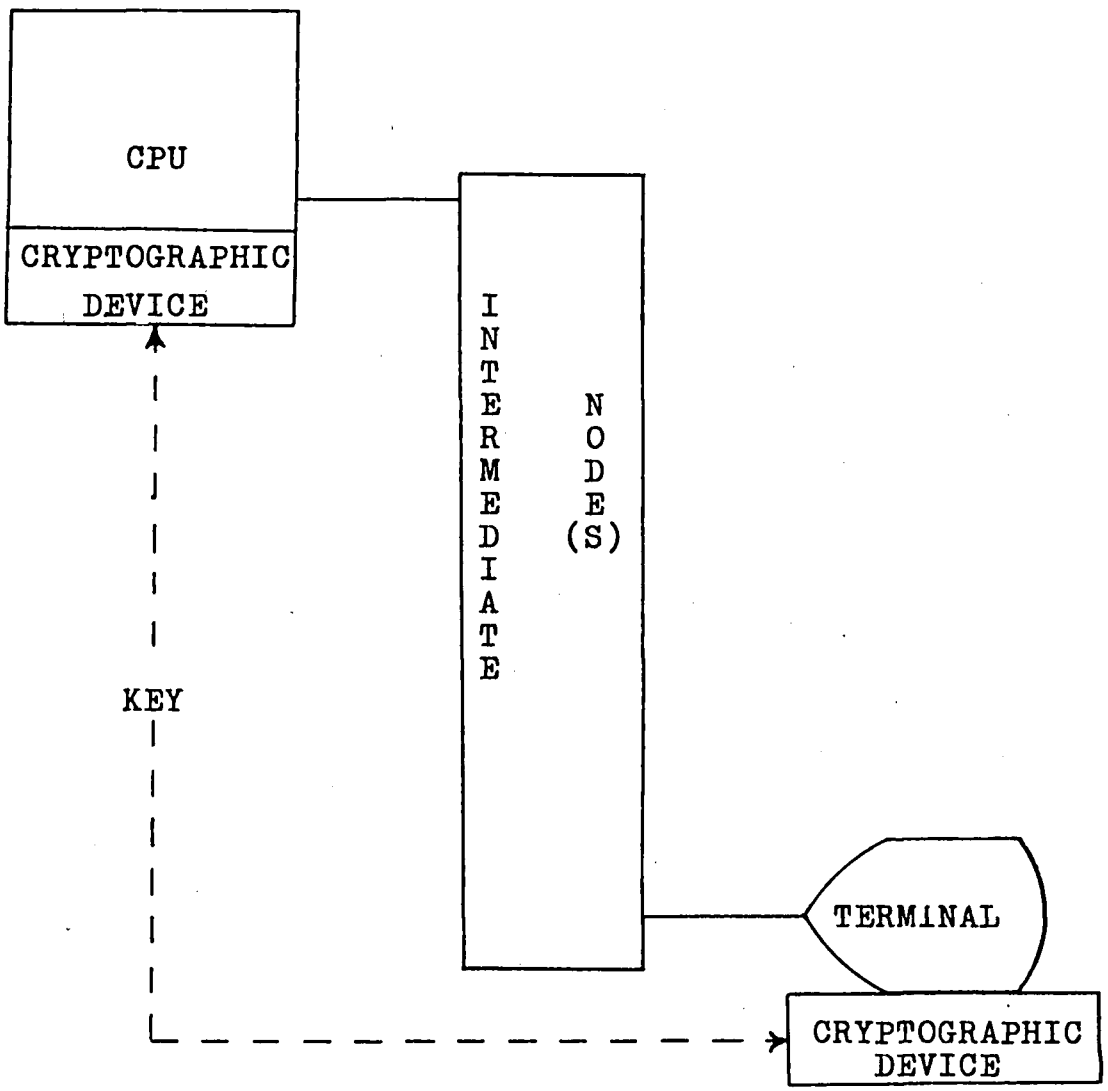


Figure 2.3

END-TO-END ENCRYPTION

2.5 THE KEY DISTRIBUTION PROBLEM

In order for a large teleprocessing network to provide privacy to all its users by implementing a conventional cryptosystem, it is first necessary for each possible pair of users to be given a key to share. The number of possible pairs U who may wish to communicate is equal to:

$$U=(N^2- N)/2 \qquad (2.1)$$

where N is the number of users [6]. If we think in terms of an electronic mail system it is easy to imagine N being on the order of the number of telephones in the country. The catch is that before a pair of users may use the key to secure an insecure channel, the key must be sent over a secure channel. Generally speaking the users will not have access to a fast secure channel but must rely on a courier. While the use of a courier might not be a matter of inconvenience or expense in situations where two parties expect to communicate a great deal over a long time, e.g., between an embassy and the State Department many private conversations occur only once between people

with no prior acquaintance. No electronic mail system that provides privacy will ever succeed if each user must wait until a key has been delivered over a slow but secure channel. The expense and delay would not be acceptable to business or private individuals.

Figure 2.4 illustrates the key distribution problem in relation to the overall flow of data/information in what is considered a conventional cryptographic system [9]. This key distribution problem is the Achilles heel of all conventional cryptographic systems including the NBS data encryption standard discussed in chapter 3. Not until the middle of this decade, with the discovery of several possible public-key cryptosystems has the means become available to overcome the key distribution problem.

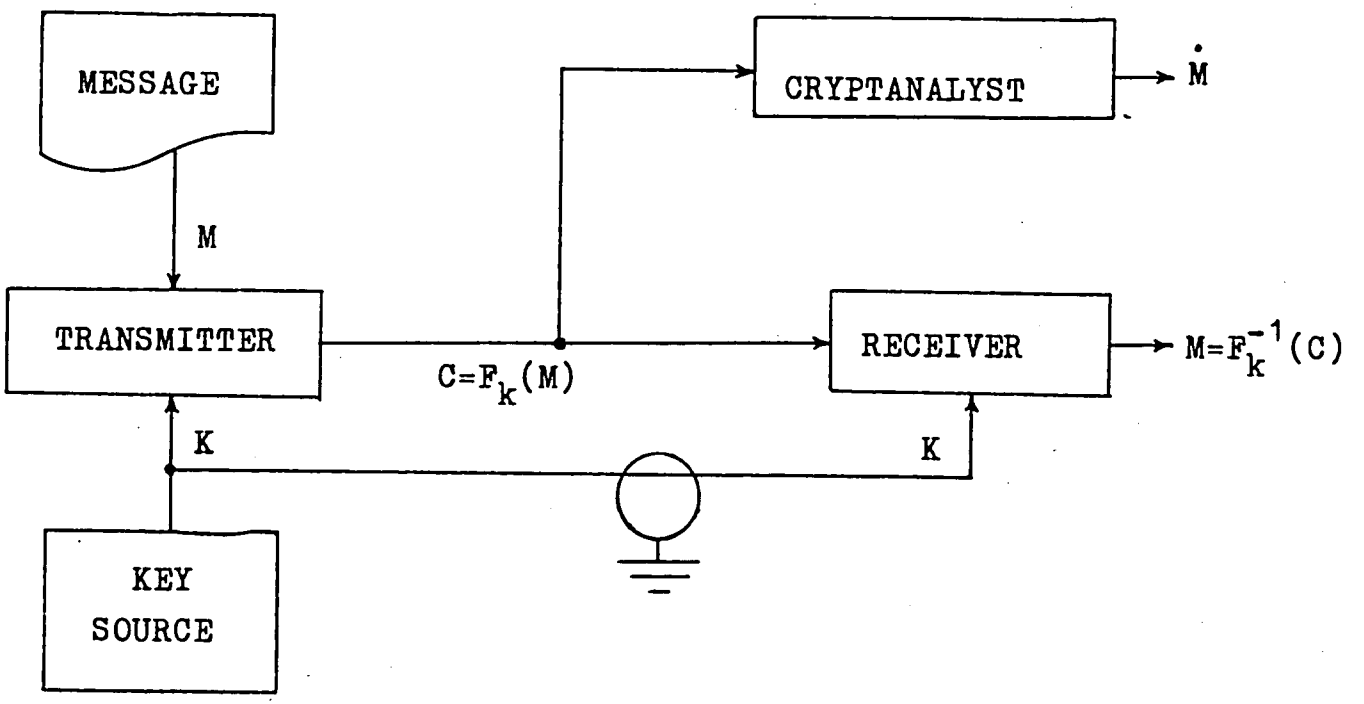


Figure 2.4

FLOW OF INFORMATION IN A CONVENTIONAL CRYPTOGRAPHIC SYSTEM

CHAPTER 3

NBS DATA ENCRYPTION STANDARD

3.1 INTRODUCTION TO THE DES

Practical applications of technological advances in the telecommunications industry, as in many other high technology industries, has been hampered by a lack of standardization. In 1971 the National Bureau of Standards (NBS) Institute for Computer Sciences and Technology (ICST) recognized a growing need for data security. The NBS concluded that data encryption is the preferred method of protecting electronically communicated data, and in some cases, for electronically stored data. To provide this cryptographic capability the NBS published in the May, 1973 issue of the Federal Register a request for the submission of encryption algorithms for computer data protection as candidates for use as a Federal Information

Processing Standard.

In the late sixties IBM had set up a cryptology research group led by Horst Feistel. This group produced an algorithm which IBM submitted to the NBS. The NBS published the algorithm March 17, 1975 and expressed their intention to have it considered as a Federal Information Processing Standard. The proposed Data Encryption Standard (DES) became a Federal standard on July 15, 1977. As a Federal standard it applies to all Federal Agencies not involved with national security work. Over the past couple of years the DES has pretty much become the de facto standard within private industry. The acceptance of the DES has been helped by the appearance in the commercial marketplace of several hardware devices from IBM, MOTOROLA, and others for implementing the DES.

The DES can be used either as a block cipher or as a stream cipher. For most applications the block cipher mode is preferred, and it is the DES operating as a block cipher which will be treated here. The DES is designed to encipher eight 8-bit EBCDIC (Extended Binary Coded Decimal Interchange Code) characters for a total of 64 bits per encryption block. There are conversion considerations when using the DES with nonEBCDIC based systems (generally

nonIBM). The enciphering is under the control of a 56-bit key. There are, therefore, 2^{56} possible keys. In general the encryption process consists of sixteen seperate rounds of encipherment; in other words the DES is a product cipher. The DES represents the state of the art in commercially available encryption. While it is public knowledge how the DES works, the why, in the form of IBM's notes, has been classified by the National Security Agency. This may be a sign of weakness in the standard.

The key consists of a total of 64 bits, 56 bits for use by the algorithm and eight bits are available for parity checking if desired. In each round of enciphering a subset of 48 bits of the 56-bit key is used. The subkeys are derived from a shifting scheme applied to the 56-bit key. The deciphering operation uses the 48-bit subkeys in reverse order from the enciphering operation.

The DES creates a strong intersymbol dependence. Each ciphertext bit is the result of a complex function of all plaintext and key bits. Changeing of a single bit of ciphertext will cause each subsequent bit of deciphered ciphertext to differ from the correct plaintext with a probability of 0.5. Ehrsam et al [9] suggest that by appending a short known bit pattern to a plaintext prior

to encryption, and comparing the known bit pattern to the decrypted pattern, the true content of the message can be checked. If block encryption is used the pattern of test bits must be appended to each block of plaintext. Using chained block encryption, the test bits need only be appended to the final block of plaintext; which will provide a significant savings in time and space for operation of the cryptosystem. Equipment currently available for implementing the DES can achieve encryption/decryption speeds on the order of two million bits a second.

3.2 CRYPTANALYSIS OF THE DES

The DES has come under attack. Diffie and Hellman [10] have claimed that a machine could be built for twenty million dollars that could break the DES, i.e., find a particular key in about twelve hours. Depreciation of the machine over five years would produce an equivalent cost per solution of about 5000 dollars. The proposed method of attack consists of nothing more than simple enumeration of all possible keys, encryption of a known plaintext, and comparison with the actual ciphertext. The proposed machine would contain approximately one million parallel

pairs of a microprocessor, for control and interfacing, and a LSI chip, for the calculations, each trying one of the $2^{56} = 7.2*10^{16}$ possible keys each microsecond. An exhausive search would require one day. Since, on average, only half the keys need be tried before the correct key is found, only half a days time is needed to break the cipher.

Both NBS and IBM have objected [11] that equipment capable of such a known plaintext attack using exhaustive search would not be available until 1990 and even then would cost 200 million dollars.

Diffie and Hellman [10] concluded that the DES should be redesigned with a 128-bit or larger key to rule out exhaustive search as a viable cryptanalytic attack. Tuchman and Meyer [11], the principle designers of the DES, have responded by concluding that the DES is secure for at least five to ten years into the future. They suggest SUPERENCIPHERMENT, the enciphering of an already enciphered message, as a means of extending the period over which the DES will remain secure. For example, if a message is enciphered twice, using two different keys, the effect is essentially the same as doubling the key size to 112-bits. This would not complicate the interconnection

of old and new equipment the way a change to a larger key would. Unfortunately superencipherment only aggravates the key distribution problem which the DES, a conventional cryptosystem, is plagued by.

Several different protocols for the generation, distribution, installation, and general management of DES keys have been developed, see Ehrsam et al [9], Matyas et al [12], Gladney [13], and Meyer et al [14]. All depend on the use of a courier or similar secure means to distribute at least key-encryption keys to the users. One suggestion of Matyas et al [12] is to use several secure channels and distribute only part of the key over each channel thereby reducing the chance of compromising the key. Some critical remarks concerning this idea appear in the next section of this chapter.

3.3 CRITICAL REMARKS ON A KEY DISTRIBUTION PROTOCOL

In light of the arguments of Diffie and Hellman it would seem unwise to proceed to use a key even if only a portion of it was intercepted since the space over which an opponent must perform an exhaustive search might be substantially reduced. In fact, the solution space is reduced by a factor of two for each key bit that is

completely known. By "completely known" is meant the cryptanalyst knows the value of the bit, whether 0 or 1, and also knows the position of the bit within the key.

For example, if an opponent intercepted one seventh of a key, i.e., eight bits, and knew which eight bits of the 56 it was he/she would have to search:

$$(2^{56} - 2^{8})/2 = 2^{48}/2 = 2^{47} \qquad (3.1)$$

bit patterns on average before the system was broken.

Another problem exists when an opponent learns information about the parity bits contained in the key. Recall that each key contains 56 key bits and eight parity bits for a total of 64 bits. Each group of seven key bits has one parity bit associated with it. If an opponent should intercept a parity bit plus knowledge of which seven bit section it was associated with he/she would be able to substantially reduce the keyspace over which an exhaustive search must be made.

As an example, suppose an opponent learns that an odd parity is being used. This implies that the parity bit for each seven bit group is adjusted so that the sum of all eight bits is odd. Suppose also that the opponent has

learned that the parity bit associated with the first seven key bits is odd (parity bit=1 implies odd parity). Then, using standard notation the cryptanalyst would know not to try the bit patterns shown in Table 3.1 for the first seven key bits since the addition of the parity bit would indicate even parity which is contradictory to what is known.

0000001
0000010
0000100
0001000
0010000
0100000
1000000

Table 3.1

All possible bit configurations with one bit on.

Similarly all the bit patterns that indicate three bits equal to 1, five bits equal to 1, and all seven bits equal to 1 can be ignored. This amounts to:

$$\binom{7}{1} + \binom{7}{3} + \binom{7}{5} + \binom{7}{7} = 7+35+21+1=64=2^6 \qquad (3.2)$$

where

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \qquad (3.3)$$

defines the number of different combinations of n things taken k at a time without repitition.

since

$$2^7 - 2^6 = 2^6 \qquad (3.4)$$

the number of keys that need be examined is

$$2^{49} \times 2^6/2 = 2^{56}/2 = 2^{54} \qquad (3.5)$$

Therefore it can be seen that complete knowledge of one parity bit is equivalent to complete knowledge of one key bit. In both cases the keyspace is cut in half. In consequence there may actually be a greater risk in sending portions of a key over several secure channels.

For example, if there is a one in fourteen chance that a courier may prove unreliable, one would expect to send fourteen keys and lose only one when there is only one courier trip per key delivery. If it is decided to break each key into seven equal parts and send each part with a different courier, i.e., over different secure channels one should only safely expect to send one key out

of every two since during one of the fourteen courier trips associated with the sending of two keys one of the couriers will be bribed, or in some fashion compromise his/her portion of the key. While it is true our opponent must still search 2^{49} keys, on average, this is a very very substantial reduction from 2^{56}. and may not be beyond the opponents means.

If it is known that a portion of a key has been compromised there is no problem, do not use the key, but if one can not expect to know when a portion of a key has been compromised it may be better, on average, sending an entire key over a single secure channel. Increasing the number of couriers per key is probably not a better key protocol. Consider that if, in the preceeding example, each key had been broken into fourteen parts and each sent over a seperate "secure" channel one would expect a portion of every key sent to be compromised. For a given system, if the threat environment, i.e., our opponents capabilities can be adequately described, and the most likely outcomes defined in terms of probabilities, then game theory would provide a means to choose a key distribution protocol which minimizes the loss (cost).

CHAPTER 4

PUBLIC-KEY CRYPTOSYSTEMS

4.1 ONE-WAY FUNCTIONS

It was pointed out in chapter 2 that the weakest point in a password system is the stored table of passwords. According to Wilkes [8], R. M. Needham implemented a password system at Cambridge in England which avoids the necessity of having such a table. It is based on what has come to be known as a one-way function. In a cipher system using such a function the encryption algorithm, which need not be complicated, is chosen such that the deciphering algorithm is computationaly difficult to compute. It should be difficult enough to preclude any form of attack other than trial and error.

Evans et al [15] have proposed a user authentication system similar in concept to Needham's. Essentially the system depends upon finding a function H which unauthorized users, or anyone for that matter, would find difficult to invert. The noninvertibility of H would be completely different from what is normally meant in mathematics when one refers to a function as "noninvertible". The normal convention is to say a function H is noninvertible if a point y is not unique, i.e., there exist distinct vectors X1 and X2 such that H(X1)=y=H(X2). This is not the type of noninvertibility that is needed. Instead, it must be very very difficult given a point y and knowledge of the function H to compute any vector X with the property that H(X)=y. Once such a H is obtained it can be applied to the users password table to produce a new table which is then stored in the system. Mathematically, when a user enters his/her proffered password P' the system computes H(P') and compares it with H(P) in the stored table. If H(P')=H(P) then, assuming H is one-to-one, P'=P, the correct password, and the user is logged onto the system. The system provides security not by keeping H and H(P) secret, but by means of the computational difficulty of determining P even if H and H(P) are known.

The problem is to find an H with several characteristics. One is that H be one-to-one, i.e., there do not exist two distinct passwords Pa and Pb such that H(Pa)=H(Pb). The problem here is that proving H is one-to-one may be as difficult as proving H has no easy inverse, a very complicated task. In addition, H should not be expensive either in time or space to compute. People quickly become impatient and will not accept a long delay between giving their password and being logged onto a system. Another human-factor consideration is that the password be short enough so that it may be easily remembered and offered to the system, e.g., typed in at a CRT terminal. The problem here is that if the set {P} only contains a few thousand items an intruder may gain access to the system by trying each possible P.

Purdy [16] suggests the use of sparse polynomials over a prime modulus as a candidate for H. Specifically, let p(x) be the polynomial given by:

$$p(x)= x^n+ a_1x^{n-1}+.....+a_{n-1}x+a_n \qquad (4.1)$$

where $n,a_1,.....,a_n$ are integers; and let Q be a large prime. U is said to be congruent V modulo Q if U-V is exactly divisible by Q. This is written $U \equiv V(\text{mod } Q)$. As

an example, $25 \equiv 1 (\text{mod } 12)$. Define H(x) as the unique number such that $H(x) \equiv p(x) (\text{mod } Q)$ and $1 \leq H(x) \leq Q$. H(x) then provides a mapping from the set {1,2,...,Q} into itself. Purdy claims that functions such as H(x) have a degree of degeneracy that does not exceed the degree n of the polynomial p(x). A highly degenerate function would produce the same value of H(P) for many distinct passwords P, an undesireable characteristic. Purdy claims that finding the roots of such a function can not be done by interative methods, such as Newton's; and that such methods are no better than trial and error. For practical values for the prime Q and the degree n a polynomial root finder would require on the order of $> 10^2$ years to crack the system assuming 10^6 machine operations per second.

While such systems provide authentication capability they offer no solution to the problem of insuring privacy. In fact, such systems will easily succomb to a strong effort at discovering a legitimate password via wiretapping.

4.2 PUBLIC-KEY CRYPTOSYSTEM DEFINED

Consider a cryptosystem in which the following three properties are found:

(a) There exists a deciphering procedure D such that when it is applied to the enciphered form of a plaintext M the result is M. Formally,

$$D(E(M))=M$$

(b) Both the enciphering procedure (algorithm and key) E, and the deciphering procedure (algorithm and key) D are easily computed.

(c) A user may publicly reveal E and not reveal an easy method to compute D.

An encryption/decryption function satisfying conditions (a)-(c) is a "trap-door one-way function". In similarity with the one-way functions described in section 4.1 it is easy to compute in one direction, but very hard to invert. However it would be trap-door because this one-wayness is only perceived. In fact, the inverse is easily computed once certain secret trap-door information is known. This trap-door information is very very difficult to discover, but is known to the system's designers who intentionally designed it into the system. Should the designer forget the trap-door information he

would be as unable as anyone to perform the inversion (deciphering).

In order for a cryptosystem to provide full authentication capability, a fourth property must belong to the system. Namely, E(D(M))=M when D(E(M))=M. In a cryptosystem with this property every message is the ciphertext for some other message and every ciphertext is a permissible message. Functions satisfying all four of these properties are called "trap-door one-way permutations".

Cryptosystems with these properties are called PUBLIC-KEY cryptosystems. A concept invented by Diffie and Hellman [6]. In operation each user would generate two cryptographic procedures E and D. The encryption procedure E would be made public D, the decryption procedure (actually only the key), would be kept secret. In order for a user A to send a user B a message, A must remove B's E from a system-public file and encrypt the message M by E(M)=C, and then send B the ciphertext C. B would then be able to read M after decrypting it with his/her secret D by D(C)=M.

It should be apparent that users of a public-key cryptosystem are able to establish private communication over an insecure channel without having to first send a secret key over a secure channel. This freedom from the key distribution problem makes a public-key cryptosystem ideal for use with electronic mail systems. Figure 4.1 illustrates the flow of information in a public-key cryptosystem. Note that instead of a shielded channel between each pair of users for the distribution of keys, there are only shielded channels between the users and the generator of the secret key D for secure transmission of D. If D can be generated in house this represents no problem at all.

To date, in all proposed public-key cryptosystems the decryption and encryption procedures share an algorithm, which may be made public with the encryption key; only the decryption key need be kept secret. It is conceivable that a public-key cryptosystem could be developed that uses two distinct algorithms, a public encryption algorithm, and a secret decryption algoritm. These could share a common public key. Such a system would be difficult to implement because of the lack of standardization. Also, generating two one-way trap-door functions based upon different algorithms which would

allow E(D(M))=D(E(M)) is probably extremely difficult.

4.3 SIGNATURES AND RECEIPTS

In order for electronic mail systems to compete with the existing paper mail system some method must be found to produce an electronic, digital equivalent to the written signature. This is necessary for buisness transactions, and other situations, e.g., a doctor prescribing drugs, where authentication is required. In addition to merely verifying that a message came from a particular sender, the system should protect against the threat of dispute.

The major disputes that may arise over a message are:

1. The sender may deny ever having sent a message when in fact one was sent.

2. The receiver may deny ever having received a message when in fact one was received.

3. The claim may be made that a message was sent when in fact none was sent.

4. The claim may be made that a message was received when in fact none was received.

Disputes 1 and 3 may arise legitimately in the case where a third party may forge and inject unauthorized messages into the system. Disputes 2 and 4 may arise legitimately in the case where a third party intercepts a message and does not allow it to continue on to its destination. Cryptology can not prevent the latter situation, that is a problem of physical security, but it can prevent the intercepted message from being understood.

To avoid dispute 1 a receiver can demand that the sender sign a message. With a public-key cryptosystem this can be accomplished by having the sender S encrypt the message to be sent, either plaintext or ciphertext, using Ds (the sender's decryption key). In this way:

$$Msigned=Ds(M)$$

or if privacy is also desired

$$Csigned=Ds(C)$$

where

$$C=Er(M)$$

or similarly,

$$Csigned=Er(C)$$

where

C=Ds(M).

The receiver R could later prove that S sent Msigned by applying the sender's public Es to it to recover M. That is,

Es(Msigned)=Es(Ds(M))=M

or in the case where the message has been encrypted

Es(Csigned)=Es(Ds(C))=C=Er(M)

where

Dr(Er(M))=M

or similarly

Dr(Csigned)=Dr(Er(C))=C=Ds(M)

where

Es(Ds(M))=M

It is apparent that Msigned must uniquely identify M since electronically a signature could be attached to any message whatsoever.

To avoid dispute 2 it is necessary for the sender to get the receiver to return a receipt in the form of the sent message encrypted using the receiver's secret decryption key. If M is sent the receiver returns Dr(M), if C is sent the receiver returns Dr(C). In court S can prove R received M or C by decrypting Dr(M) or Dr(C) using R's public Er to produce M or C, the message S claims R to

have received.

In dispute 3 the burden of proof is on the supposed sender to produce a signed receipt as described above. The sender will be unable to forge a receipt not knowing Dr.

In dispute 4 the burden of proof is on the supposed receiver to produce a signed message from the sender who is claimed to have sent it. The receiver will be unable to forge the senders signature, not knowing Ds.

In cases where a user allows his/her D to be compromised a court will probably allow a grace period of a reasonable amount of time in which the user could report the loss, after which the user would become liable for damages caused by the unauthorized use of D. This would be analogous to the situation existing with credit cards.

4.4 THE RSA PUBLIC-KEY CRYPTOSYSTEM

At present there are only a few contenders for a public-key cryptosystem. The trap-door knapsack system of Merkle and Hellman [17], and the system invented by Rivest, Shamir, and Adelman [18] (henceforth referred to as the RSA system) have received the most attention.

The trap-door knapsack system relys on the NP-completeness of the famous knapsack or cargo loading problem. In this system each user U deposits in a public file a trap-door knapsack vector A(U). When someone wishes to send the Uth user a message in the form of a vector X, he/she reads A(U) from the public file and computes S=X*A(U). Only the intended recipient can recover X from S.

NP (nondeterministic, polynomial) problems are solvable in polynomial time on a computer with an unlimited degree of parallelism. It is relatively easy to check a guessed solution to an NP problem of size n, the number of computational steps, and therefore the time required increases proportionaly to a polynomial in n, e.g., n^2+n^3. The best method for finding the correct solution, however increase in time much more rapidly as n grows, usually proportional to an exponential function in n, e.g., 3^n+2^n. Problems in the complexity class NP are good candidates for constructing trap-door one-way functions.

Since the primary task of this thesis is to write a set of FORTRAN programs for encrypting and decrypting messages by the RSA method using the DECsystem-20

computer, and to use these to investigate a proposed cryptanalytic attack of the RSA system, space will be taken here to fully explain the RSA cryptosystem. This discussion is taken in part from [18].

The RSA system uses a public encryption key E=(e,n) and a secret decryption key D=(d,n), d, e, and n are positive integers. The message M to be encrypted must first be represented as a number between 0 and n-1. M=0 and M=1 will be encrypted into themselves. It may be necessary to break the message into a series of blocks. To encryot M raise it to the e-th power modulo n. Formally.

$$C \equiv E(M) \equiv M^e \pmod{n} \qquad (4.2)$$

Decryption is achieved in like manner replacing M with the ciphertext C, and e with d. Formally.

$$M \equiv D(C) \equiv C^d \pmod{n} \qquad (4.3)$$

The encrypted form of a message or block of a message will also be in the range 0 to n-1. The n is the product of two very large randomly selected prime numbers P and Q.

The security of the system resides in the fact that while it is easy to compute the product of two numbers it is very difficult to decompose a given number; into its prime factors. This difficulty of factoring n hides P and Q and hence the way d can be computed from e. Rivest et al in [18] include a table, which is reproduced here, of the time required to prime factor numbers as the length (number of digits) grows. The values in table 4.1 are based upon what Rivest et al consider to be the fastest factoring algorithm known.

DIGITS	NUMBER OF OPERATIONS	TIME
50	1.4×10^{10}	3.9 hours
75	9.0×10^{12}	104 days
100	2.3×10^{15}	74 years
200	1.2×10^{23}	3.8×10^{9} years
300	1.5×10^{29}	4.9×10^{15} years
500	1.3×10^{39}	4.2×10^{25} years

Table 4.1

Factoring time based on one operation per microsecond.

The integer e is picked from a large enough set to preclude its discovery by trial and error. It is chosen to satisfy

$$gcd(e,(P-1)*(Q-1))=1 \qquad (4.4)$$

where "gcd" is an abreviation for greatest common divisor. Any prime number greater than max(P,Q) will do. Experience indicates that if a smaller number is desired it does not require more than a few trials before an e satisfying the above relation is found.

The integer d is computed from P, Q, and e so as to be the multiplicative inverse of e, modulo((P-1)*(Q-1)). That is

$$e*d \equiv 1 (mod (P-1)*(Q-1)). \qquad (4.5)$$

The correctness of choosing d in this fashion can be demonstrated using an identity due to Euler and Fermat. For any integer M which is relatively prime to n, i.e. n and M have no common factors other than 1, $M^{@(n)} 1 (mod\ n)$. Where @(n) is Fuler's totient function which defines the number of positive integers less than n which are relatively prime to n. In the case of a prime number P,

$\phi(P)=P-1$ By elementary properties of the totient function:

$$\phi(n)= \phi(P)* \phi(Q)$$

$$\phi(n)=(P-1)*(Q-1)$$

Because e is relatively prime to the totient function of n, it has a multiplicative inverse in the ring of integers modulo $\phi(n)$ and equation 4.5 holds.

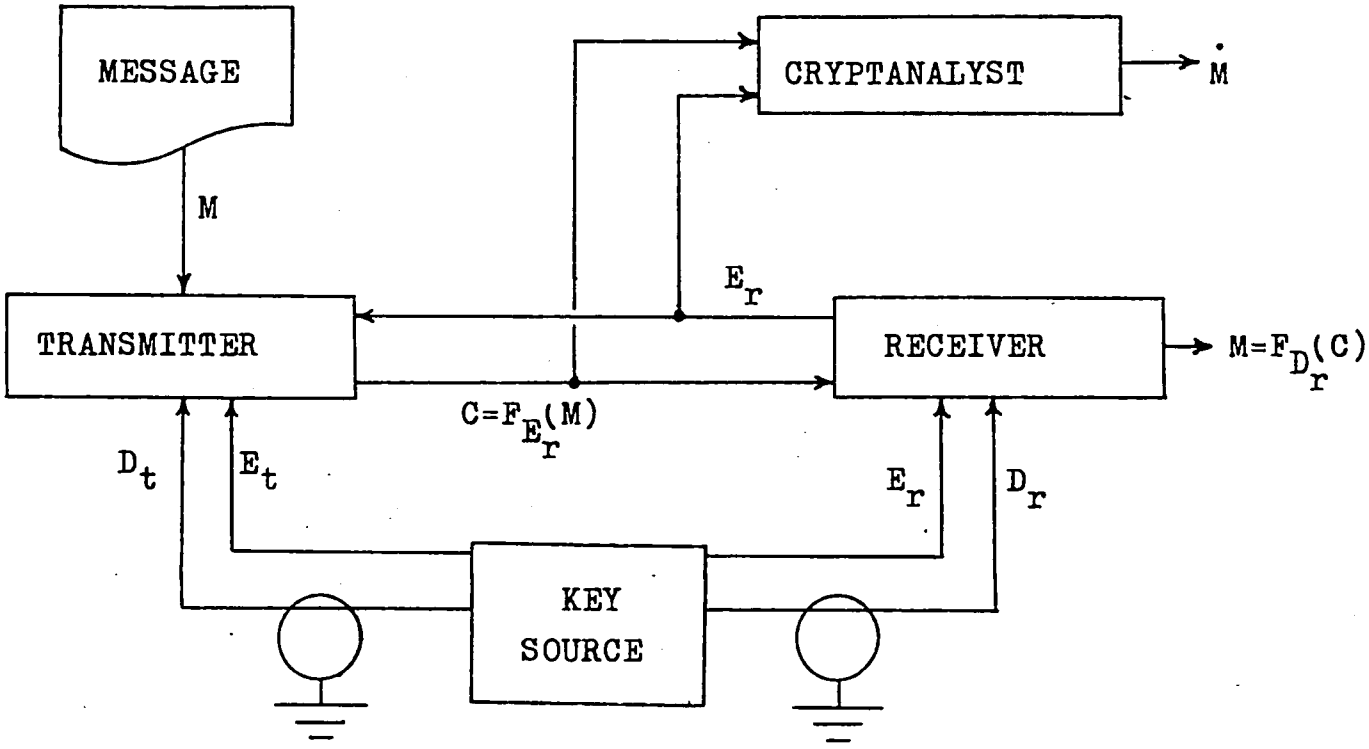


Figure 4.1

FLOW OF INFORMATION IN A PUBLIC-KEY CRYPTOSYSTEM

CHAPTER 5

PROGRAMMING THE RSA PUBLIC-KEY CRYPTOSYSTEM

5.1 MULTIPLE-PRECISION OPERATIONS

In [18] Rivest states, "an 80-digit n provides moderate security against an attack using current technology...". The goal chosen, therefore, was to develope an encryption/decryption program that could utilize at least an 80-digit n. Arithmetic on 80-digit numbers is beyond the working precision of all commercially available computers; even beyond the the double-precision routines available on such large word size machines as the CDC 6400. The DECsystem-20 computer was chosen as the machine on which the public-key cryptosystem would be developed. The reasons for this were:

1. The author was familiar with it

2. The excellent interactive operating system.

3. The existing electronic mail capability.

4. It is a more conventional system than the other main computer available at Lehigh University, the CDC 6400.

While the CDC 6400 would provide much greater speed of execution, its poor interactive operating system (at Lehigh) plus the fact that the entire ASCII character set can not be used ruled it out as the machine for this project.

The DECsystem-20 allows integers to be represented within the range of (-2**35)+1 to (+2**35)-1 (-34359738367 to +34359738367). This dictates that multiprecision arithmetic operations be programmed. Double-precision on the DECsystem-20 would not simplify things, because double precision constants are similar to real constants, and secondly, depending upon their magnitude there is only precision of 16 to 18 places. Pure integer double-precision is unfortunately not available. In any case, double-precision operations require nearly twice the time of single-precision operations. The time savings

would not be as great as one might assume.

In order to implement the RSA cryptosystem it was necessary to program all the conventional arithmetic operations (addition, subtraction, multiplication, division, and modulo) in multiple-precision for positive integers. In general the classical algorithms were used. Flowcharts of these algorithms appear in APPENDIX I. Programs were written in FORTRAN because it is a very "portable" language, unlike assembler or machine language. In the programs a multiprecision variable or constant is stored in a FORTRAN dimensioned array. Each array element stores 4 digits (if possible) of the value of the variable or constant. For example, the number 1,234,567,890 would be stored in the elements of the array N as:

N(1)=7890 N(2)=3456 N(3)=0012

Addition, modulo, subtraction, and division would have benefited from storing more digits per array element (the DECsystem-20 permits the storage of all possible integers up to 10 digits long in one FORTRAN integer array element, or constant), but the multiplication algorithm used could produce an overflow condition if more than 4 digits are stored. In addition an integer variable must be maintained that indicates how many array elements are used by a number, to prevent unnecessary operations on empty

(equal to zero) array elements.

Knuth [19 Chapter 4] offers several algorithms which would execute in significantly less time, but these can only practicaly be implemented when programmed in machine or assembler language on a bit addressable processor. These algorithms are designed to take advantage of the binary arithmetic used by the processor. Unfortunately the binary representations of a number requires about 3.4 times as many digits as the base 10 representation. Since with a high level language such as FORTRAN the registers, accumulator, and condition flags are not directly addressable to perform binary arithmetic, numbers must be converted to their binary representations with each digit stored in a seperate variable or array element. This is a highly inefficient use of storage. The binary arithmetic operations would have to programmed; and variables would have to be set aside to act as the condition flags (carry, zero, etc.). In essence a FORTRAN simulation of a binary processor would have to be developed to implement the fast algorithms in FORTRAN. This would be highly inefficient. Assembler language was not used since it is much more machine dependent than FORTRAN.

5.2 PRIME NUMBERS

The RSA cryptosystem requires the capability of generating large prime numbers. Recall that n in the encryption process $C \equiv M^e \pmod{n}$ and in the decryption process $M \equiv C^d \pmod{n}$ is the product of two primes P and Q. The security of the system resides in the difficulty of determining P and Q from n. P and Q should not be discoverable by direct search over the known primes. Consequently both P and Q should be large.

Fortunately the problem of finding large primes (with reasonable certainty of primality) is nowhere near as difficult a problem as prime factoring a number. Tables of prime numbers exist which contain all the primes less than 10^9, but for larger numbers one must resort to tests for primality.

The number of primes is infinite, unfortunately the distribution of primes follows no apparent pattern. The number of primes less than some number N as N approaches infinity is derived from the formula:

$$\lim_{N \to \infty} \frac{\pi(N) \ln N}{N} = 1 \qquad (5.1)$$

which is the prime number theorem. From this it is possible to approximate the expected number of numbers Z between consecutive primes for some arbitrary point in the integers N as.

$$\ln N=Z \tag{5.2}$$

Since all prime numbers other than 2 are odd, 50% of Z may be discarded automatically. Another 10% consisting of all the integers ending in a 5 may also be discarded since any integer ending in 5 is divisible by 5. Therefore, only 40% of Z need be checked before one would expect to find a prime. For example, for n=10 Z*.40=10, and for n=10 Z*.40=92.

It is possible to prove that a number P is prime by the following converse to a theorem by Fermat.

THEOREM: If there exists a number X for which the order of X modulo P is equal to P-1, then P is prime.

The order of X modulo P is the smallest positive integer k such that $X^k \bmod P=1$. The order of X will be P-1 iff:

1. $X^{P-1} \bmod P=1$ and,

2. $X^{P-1/Q} \bmod P=1$ for all primes Q which divide P-1.

Unfortunately it is necessary to know the prime factors of P-1. This can easily be a severe limitation. It is necessary then to look for a method that will identify a number as prime with a probability high enough for the application at hand.

Rivest et al [18] suggest a "probabalistic" algorithm due to Solovay and Strassen [20 and 21]. This algorithm tests a number P for primality by testing whether:

$$\gcd(A,P)=1 \quad \text{and} \quad J(A,P)=A^{P-1/2} \pmod P \qquad (5.3)$$

where A is randomly chosen from a uniform distribution on {1,...,P-1}, and J(A,P) is the Jacobi symbol. If P is prime it will always satisfy the conditions given above. If P is composite it will falsely satisfy the conditions with a probability at most .5. Rivest et al suggest that 100 random values for A be tried. If P passes the test for all 100 then there is a chance of only one in 2^{100} that P is not prime. This algorithm was not used for two reasons. First, because of the difficulty of calculating

J(A,P) in a reasonable amount of time, and secondly, because a reliability of one in 2^{100} is unnecessary.

The method chosen for determining if a number P is prime consisted of five steps. First, all even numbers and numbers ending in a 5 are not even considered. Secondly, the gcd(P,A_i) is calculated as in Solovay's and Strassen's method. This was modified so that instead of 100 random A_i the A_i were chosen so that as i=1,2,...,5 the A_i are 8 or 7 digit integers which are the products of all prime numbers less than 103 (except 2) used only once, i.e., each A_i uses a subset of the primes less than 103 as its factors. The problem of fitting j factors X_j into i products A_i so that each X_j is used only once and each A_i does not exceed some maximum value can be treated as a knapsack problem and is easily solved by recalling that the log of a product is equal to the sum of the logs of its factors. This step quickly eliminates integers with small prime factors. Lehmer [22] determined that 90% of the 90 million numbers between 10^7 and 10^8 have prime factors < 313. Of course Lehmer's range included the even integers all of which possess 2 as a prime factor and integers ending in a 5, all of which possess 5 as a prime factor. The third step was to calculate gcd(P,A_i) where i=6,7,...,95 and the A_i are randomly selected on the range

{1,...,P-1}. If a number is prime its gcd with any number less than itself is always 1. According to a theorem by E. Cesaro:

THEOREM: If P and A are integers chosen at random, the probability that gcd(P,A)=1 is $6/\pi^2$.

While by this step P cannot be considered randomly selected, and the probability of $gcd(P,A_i)=1$ is probably greater than $6/\pi^2$, one should expect that after being tested against 95 A_i P is composite with low probability. The fourth step utilizes the previously stated converse to the theorem of Fermat. Mathematically when P is not prime, it is always possible to find an A<P such that A mod P=1. Nearly 25 centuries ago the Chinese discovered that for the case A=2, if 2 mod P=1 then P is prime. Unfortunately this is wrong, there are exceptions, but they are rare. For example, of all the numbers less than 50 million only 1511 satisfy the test which are not prime. This is very significant if one considers that there are approximately 3 million prime numbers less than 50,000,000. The fifth step consists of performing the calculation of the fourth step using four different values of A to, hopefully, eliminate candidates for primality

that pass step four.

One could probably acheive satisfactory results using just steps four and five, but since these require significantly more time to calculate than the gcd the preliminary elimination is desireable.

5.3 PROGRAM PRIME

Program PRIME is the FORTRAN program written to find large prime numbers by the five steps given in section 5.2. Copies of PRIME, as well as all programs written for this thesis are stored in the Industrial Engineering Department at Lehigh University.

The calculation of gcd(P,A_i) was accomplished using Euclid's algorithm which for positive integers consists of two steps:

1. If A divides P, STOP with A as the answer.
2. If P mod A is equal to 1 A and P are relatively prime so STOP, otherwise make P=A and A=P mod A and return to step 1.

The calculation of A^{P-1} mod P is essentially the same as what is required for the RSA encryption/decryption process. The effecient technique used is one called "exponentiation by repeated squaring and multiplication". The steps are:

1. Let $p_k p_{k-1} \ldots p_1 p_0$ be the binary representation of P-1.

2. Set the variable C to 1.

3. For i=k,k-1,.....,0. set C=C mod P and if p =1 set C=(C*A) mod P.

4. STOP now $C=A^{P-1} \bmod P$.

The reliability of PRIME was tested in two ways. First the program was run to generate all the prime numbers between 10,000,439 and 10,006,721. The result was checked against Lehmer [23]. PRIME identified all 400 primes in the range and did not falsely identify any composite number as prime. This run required approximately 3.5 minutes of CPU time. Second, the first 50 entries in Lehmer's [22], "Table of Composite Solutions n of Fermat's Congruence $2^n \equiv 2 \pmod n$ and Their Smallest Prime Factor p" were checked. PRIME correctly identified each of these

numbers as composite. This test was used to determine that step 5 of the selection process be done for 4 A in addition to 2. It takes approximately half a minute of CPU time to find a 40-digit prime using PRIME.

5.4 PROGRAM NEXTPR

Rivest et al suggest that to afford additional protection against sophisticated factoring algorithms, the factors of n, i.e. P and Q, should differ in length by a few digits, and P-1 and Q-1 should each possess at least one large prime factor. In addition, gcd(P-1,Q-1) should be small. Finding a prime number P such that P-1 has a large prime factor is not difficult. Program NEXTPR accomplishes just that using the prime identifying logic of program PRIME, and the suggestion in [18] to generate a prime number U, and then let P be the first prime in the sequence (i*U)+1, for i=2,4,6,8,....., and similarly for Q. NEXTPR can do this in about a minute of CPU time for a 40-digit U. One may find it necessary to try several values for U before a P the desired number of digits is found.

5.5 PROGRAM EFIND

Determining e (actually whether one starts by choosing e or choosing d is a matter semantics since the RSA system relys on a one-way trap-door permutation function) is accomplished by finding a number relatively prime to @(n). The smaller the e, the larger the d will be and conversly. Recall though that d should come from a set large enough to preclude discovery by direct search.

Program EFIND calculates the first number e, which is relatively prime to @(n), after some user supplied guess for e. @(n) is determined by the P and Q supplied by the user. EFIND allows the user to specify a lower bound on e. Euclid's algorithm as outlined in section 5.3 is used to calculate the gcd of @(n) and each value of e tried by the program until the gcd=1; implying relative primality.

5.6 PROGRAM KEY

Program KEY accepts as input a U and a P (found using PRIME and NEXTPR), and an e (found using EFIND) and generates a decryption exponent d, and n=P*Q. KEY stores d and n in a decryption key file specified by the user, and e and n in an encryption key file also user named.

The key files are set up by KEY to be used by program RSA, the encryption/decryption program.

The calculation of d is accomplished using a variation on Euclid's algorithm for computing the gcd presented earlier. The program calculates gcd(@(n),e) by computing a series $x_0, x_1, x_2 \ldots$, where $x_0 = @(n)$, $x_1 = d$ and $x_{i+1} \equiv x_{i-1} \pmod{x_i}$, until an x_k equal to zero is found. Then $\gcd(x_0, x_1) = x_{k-1}$. A series of numbers b_i are computed by $b_i = b_{i-2} - t * b_{i-1}$ for $i>1$, where t is the quotient produced during the calculation of $\gcd(x_{i-2}, x_{i-1})$. To begin $b_1 = 1$, $b_0 = 0$. The b_i are calculated so as to provide a solution to:

$$x_i = a_i * x_0 + b_i * x_1 \qquad (5.4)$$

Where the a_i can be calculated similarly to the b_i. Note that it is not necessary to calculate the a_i. When $x_k = 0$, x_{k-1} equals 1 and b_{k-1} equals d, the multiplicative inverse of e, modulo @(n). The program produces an error message if the d found is negative, in which case another e or @(n) should be tried. This calculation requires less than a minute of CPU time for n<80-digits. One final note: if the d found or e chosen is less than $\log_2(n)$ not every message will be reduced modulo n. A quick idea of the

magnitude of e and d required can be had by using a pocket calculator, and the following relation if logs to the base 2 can not be gotten directly on the calculator used.

$$\text{Log}_2(n) = \frac{\text{Log}_{10}(n)}{\text{Log}_{10}(2)} \qquad (5.5)$$

In a working system the encryption key file would be stored in a public directory and the decryption file would be stored in the user's private directory. On the DECsystem-20 a directory is a named collection of files. A file is a named collection of records containg either program code, text, data, etc. A private directory is only accessable by those who know the correct password. A public directory is accessable by anyone who can log onto the system. All programs necessary for key generation and encryption/decryption would be stored in a public directory.

5.7 PROGRAM RSA

Program RSA is the encryption/decryption program. RSA can encrypt any text file composed of the ASCII characters given in table 5.1. This set contains all the text characters on a standard ASCII keyboard. RSA prompts

the user to specify whether encryption or decryption is

@=01	S=20	f=39	y=58	-=77
A=02	T=21	g=40	z=59	.=78
B=03	U=22	h=41	{=60	/=79
C=04	V=23	i=42	|=61	0=80
D=05	W=24	j=43	}=62	1=81
E=06	X=25	k=44	~=63	2=82
F=07	Y=26	l=45	=64	3=83
G=08	Z=27	m=46	!=65	4=84
H=09	[=28	n=47	"=66	5=85
I=10	\=29	o=48	#=67	6=86
J=11	]=30	p=49	$=68	7=87
K=12	^=31	q=50	%=69	8=88
L=13	_=32	r=51	&=70	9=89
M=14	`=33	s=52	'=71	:=90
N=15	a=34	t=53	(=72	;=91
O=16	b=35	u=54	)=73	<=92
P=17	c=36	v=55	*=74	==93
Q=18	d=37	w=56	+=75	>=94
R=19	e=38	x=57	,=76	?=95

Table 5.1

ASCII characters and their assigned numerical values.

desired, then requests the specification of the file containing the appropriate key. If encryption is desired the user is requested to supply the specification of the file containing the plaintext. RSA encrypts and decrypts message files one record (block) at a time. Since for encryption each ASCII character is converted to a two digit number (see table 5.1) there is a 2 to 1 expansion of the text. Because of this it may be necessary to reblock the plaintext file (change the number of plaintext

characters per record) so that when each record is converted to a number the result is less than n. After reading the encryption key file RSA calculates the maximum number of characters (left justified) per record that can be encrypted based upon the n in the key file. The user must specify the number of characters per record to be encrypted. If the user's text file contains more characters per record than may be encrypted based upon the n in the key file the user must stop execution and reblock his/her text file. Finally RSA requests the user to supply the ciphertext file specification of the file into which the encrypted form of the message will be written.

The first record of the ciphertext file contains the number of ASCII characters per record of the plaintext file. This information is needed by RSA to insure an exact reproduction of the plaintext file during decryption. Each remaining record contains a 2-digit number that specifies the length (number of array elements required) of the ciphertext number that is the encrypted form of a given plaintext record. These ciphertext numbers follow the lengths after two blank spaces in the record. Note that the ciphertext is not converted back to its alphanumeric version, there is no point.

During decryption the user is requested to supply the specification of the decryption key file, the ciphertext file, and a file into which RSA can write the deciphered message. See APPENDIX II for a sample execution of RSA and the other programs.

RSA can encrypt messages using an n up to 160-digits long. This implies that 80 characters of text per file may be encrypted. A plaintext or ciphertext file may contain any number of records, the only limit being the disk space alloted for a user's directory.

In order to test the speed and reliability of RSA the test plaintext file shown in table 5.2 was used.

RECORD	PLAINTEXT
1	GENESIS 11:7 Go to, let us go down, and
2	there confound their language, that they
3	may not understand one another's speech.
4	@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcde
5	fghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-.
6	/0123456789:;<=>? It's all Greek to me.

Record start Record end

Table 5.2

Test plaintext file.

This file consisted of 6 records of 40 characters each (blanks are considered characters) for a total of 240 characters. The file was reblocked for tests with different size n. Table 5.3 gives the number of characters per record and the number of records per file for the nine versions of the test plaintext file used.

CHARACTERS/RECORD:	4	6	8	10	12	16	20	30	40
RECORDS/FILE:	60	40	30	24	20	15	12	8	6

Table 5.3

Dimensions of test plaintext files.

RSA encrypts and decrypts by using the efficient algorithm for raising a number to an integer power and reducing the result modulo some integer given in section 5.3. In prime the algorithm was used to calculate $A^{P-1} \bmod P$, in RSA A is replaced by M or C, the numerical verions of the plaintext or ciphertext respectively, P-1 is replaced by e or d, depending on whether encryption or decryption is desired, and (mod P) is replaced by (mod n). The beauty of the algorithm is that since the reduction modulo n is carried out while the power is being evaluated C (in the algorithm) never grows larger than twice the

length of n.

When a record of the plaintext file is read each character is stored in a seperate array element. In order to convert a character to its 2-digit numerical equivalent and vice versa, a simple binary sequential search is performed using the computer's internal ordering of the different characters. Blanks were assigned the value 64 rather than 00 because that is where they fit sequentially as ordered by the DECsystem-20 computer, but more importantly, because making a blank equal to 00 would result in loosing information about the number of leading blanks in a plaintext record.

In order to test the execution time of RSA, two tests were devised.

1. Using increasing values of n, determine the encryption time necessary to encipher the 240-character test plaintext file (reblocked for each n) as e is held constant. The n will be similar, i.e., having the same leading digits.

2. For the test plaintext file using 20-character blocks and a constant n, determine the encryption time for increasing values of e.

These two tests provided results which will allow one to determine a good approximation to the execution time of RSA for a given block size and size of e. The results are also applicable to the decryption process since the same type of calculations are performed.

For the first experiment nine P and Q pairs had to be found whose products, i.e., the different n, would have similar leading digits. Similar leading digits were desired to eliminate variations that might occur using different leading digits. The leading digits of n govern how the division proceeds during the reduction modulo n of the encryption/decryption process. The P and Q pairs were chosen so that the 9 n produced were each one digit longer than two times the number of characters/record given in table 5.3. An n that is twice the block size digits long could be used, but if just four more characters are added to the character set and assigned the remaining two digit values of 96,97,98,and 99 it would be possible to have a plaintext record that would convert to a string of nines. Using an n one digit longer than twice the block size eliminates any restrictions on the values of any of the digits of n. The P and Q pairs and the products n along with the prime factorizations (where known) of P-1 and Q-1 are given in table 5.4.

For 4 characters/record:
P=982063
P-1=2*3*3*54559
Q=557
Q-1=2*2*139
n=547009091

For 6 characters/record:
P=98765531
P-1=2*5*9876553
Q=54539
Q-1=2*27269
n=5386573295209

For 8 characters/record:
P=9876543191
P-1=2*5*987654319
Q=5432411
Q-1=2*5*543241
n=536534418727635O1

For 10 characters/record:
P=987654321371
P-1=2*5*98765432137
Q=543202691
Q-1=2*5*54320269
n=536496485146506009361

For 12 characters/record:
P=98765432109431
P-1=2*5*9876543210943
Q=54321090767
Q-1=2*27160545383
n=5365046002258377627723577

Table 5.4

P and Q pairs, the prime factorizations of P-1 and Q-1 and the products n.

For 16 characters/record:
P=9876543210987636l1
P-1=2*5*98765432109876361
Q=543210987662831
Q-1=2*5*54321098766283
n=536504679233522206919208640042741

For 20 characters/record:
P=98765432109876543210 67
P-1=2*3*1646090535164609053511
Q=5432109876543211691
Q-1=2*5*543210987654321169
n=536504679225118424904172990415622619942 97

For 30 characters/record:
P=987654321098765432109876543211 79
Q=543210987654321098765432109 79
n=536504679225118355439658624210641607999696692912452522502
4241

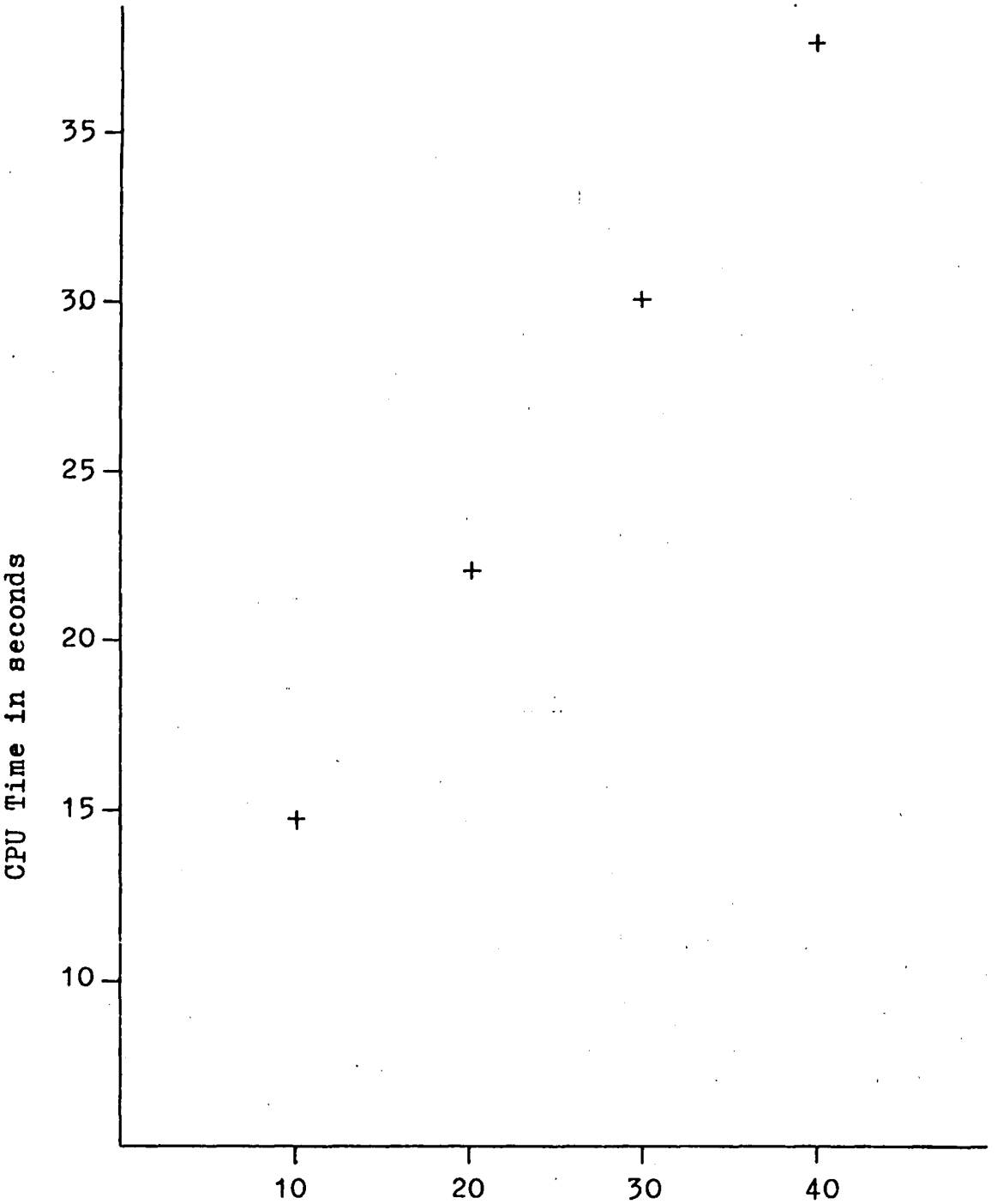
For 40 characters/record:
P=9876543210987654321098765432109876543210 83
Q=543210987654321098765432109876543210121
n=536504679225118355439658624295680540448776072929931260504
24141137154896966928104 3

Table 5.4 cont.

Examination of the algorithm used to accomplish the exponentiation and reduction modulo n indicates that each time a digit of the binary representation of e is equal to a one there is a multiplication by M and a reduction modulo n that does not occur when a binary digit of e equals zero, and similarly for d. One would expect that an e or d with a binary representation with a low density

of ones would cause RSA to execute significantly faster than an e or d of similar or possibly smaller magnitude that possesed a high density of ones in its binary representation. Taking this into consideration the e chosen for experiment 1 was picked to have roughly an equal number of zeroes and ones. The e chosen was 178,956,847 which in binary is 10101010101010101000101111.

Table 5.5 contains the results of experiment 1 for ecryption using the values of P, Q, and n given in table 5.4 using e=178,956,847, and also for decryption using the various values of d found by program KEY. The CPU times are given as MINUTES:SECONDS.TENTHS OF SECONDS. Figure 5.1 is a plot of the values in table 5.5 for encrypting 10, 20, 30, and 40 characters of text. Note the linear relationship. A simple regression equation fitting these points is Y=.7656X+6.96 where Y is the estimate of the CPU time and X is the number of characters per record. The R value for this equation is greater than .99. This equation is for encrypting a 240 character file. Divide Y by 240 to get a time per character. Note that the decryption times increase very rapidly. This is due to the fact that since e is held constant the values of d must grow as n grows if e*d divided by (P-1)*(Q-1) is to



Encrypting 240 characters with constant exponent e=178956847 and similar leading digits in n.

CPU TIME .VS. BLOCK SIZE

Figure 5.1

have a remainder of 1. All times in table 5.5 are average values based on five executions of RSA for each value of n.

	CPU TIME	
CHARACTERS/RECORD	ENCRYPTION	DECRYPTION
4	9.74	10.60
6	11.64	15.39
8	13.34	23.45
10	14.67	31.70
12	15.47	40.24
16	18.61	1:03.24
20	22.14	1:37.68
30	30.03	3:26.57
40	37.56	5:48.43

Table 5.5

Results of experiment number 1.

Additional tests indicate that if P and Q are chosen to produce an n which has a length twice the block size the execution time of RSA is about two thirds of the values given in table 5.5, but recall the caveat concerning this given earlier.

For experiment 2 the values of e chosen are given in table 5.6. They were chosen so as to provide an upper bound on the CPU time for decimal values of e and d which have the same number of digits in their binary representations, i.e., the e values chosen convert to a

string of ones in binary. For example, any e which has a binary representation 7 digits long or less will require less CPU time than 127 (binary=1111111).An e equal to 129 (binary=10000001) while larger than 127, and longer in binary will take less CPU time to encrypt a message because of the sparsity of ones in its binary representation. These e values were used with the P, Q, and n for a block size of 20 given in table 5.4.

	CPU TIME	
VALUE OF e	ENCRYPTION	DECRYPTION
$2^3 - 1$	2.80	26.60
$2^9 - 1$	4.24	27.28
$2^{15} - 1$	5.72	26.64
$2^{19} - 1$	6.61	26.30
$2^{21} - 1$	7.17	26.93
$2^{27} - 1$	8.59	26.59
$2^{31} - 1$	9.49	25.61
$2^{45} - 1$	12.98	26.18

Table 5.6

Results of experiment number 2.

Note the interesting result that each e used produced a d which used about the same amount of time for decryption. All the d values were on the order of 10^{40}. The reason for this is unknown. One would expect the d to decrease once e grows to near the square root of @(n). These results indicate that the execution time of RSA increases linearly, but discontinuously as the value of e or d increases for a given P, Q, and n. Figure 5.2 is a plot of the data in table 5.6 for encryption time. Figure 5.3 illustrates the general effect of the size of e or d on execution time.

In conclusion one should use an n the same number of digits as twice the block size unless text characters are assigned all the numbers from 1 to 99, in which case n will have to be a least one digit longer. The d should come from a large enough set to avoid discovery by trial and error, and both e and d should have mostly zeroes in their binary representations for rapid execution.

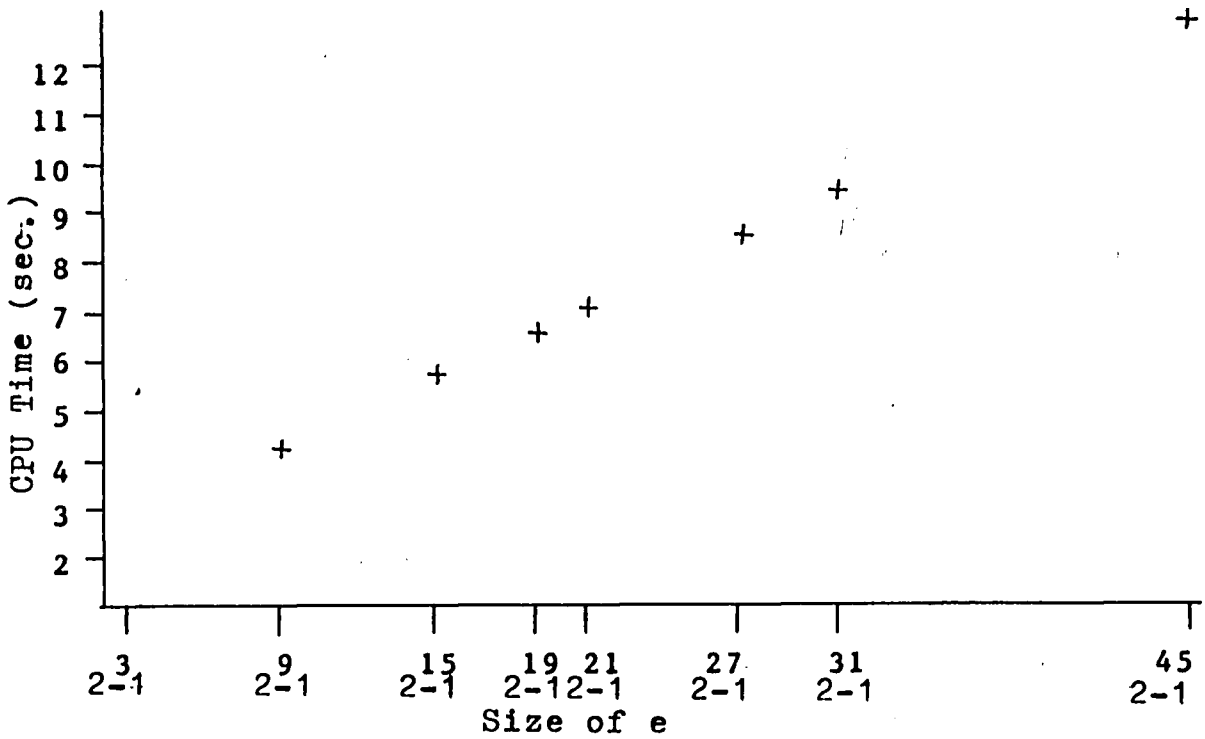


Figure 5.2

PLOT OF ENCRYPTION TIMES IN TABLE 5.6

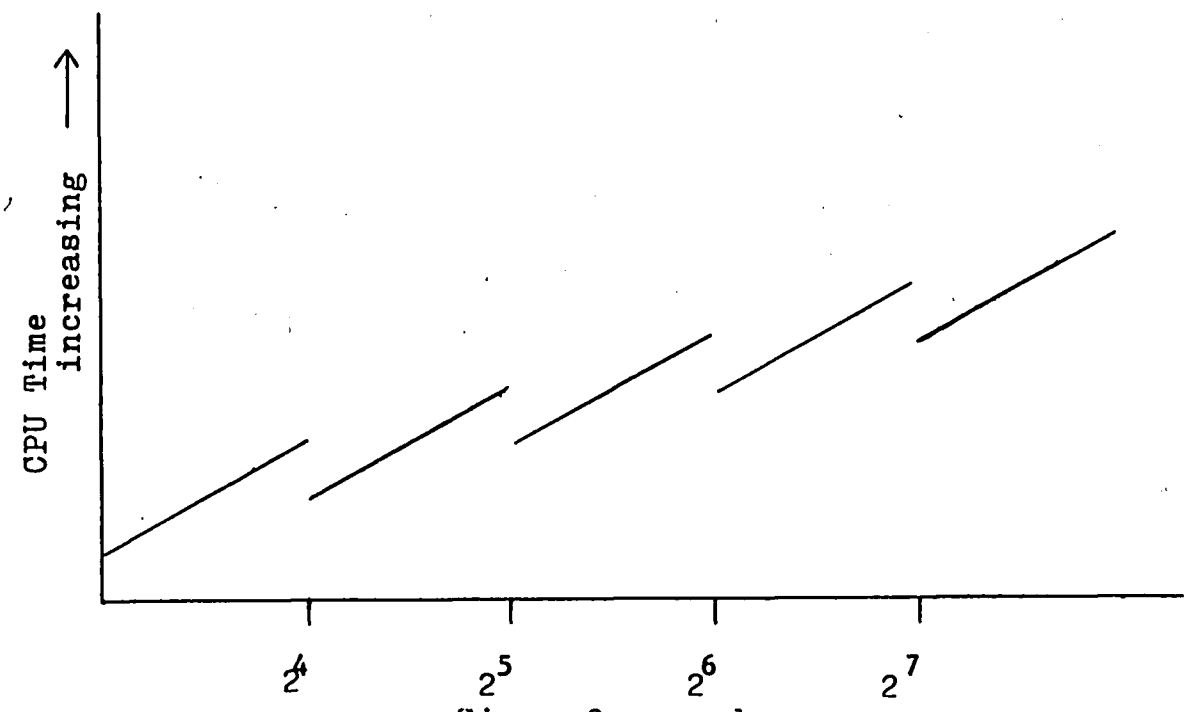


Figure 5.3

GENERAL EFFECT OF THE SIZE OF e OR d ON CPU TIME

CHAPTER 6

EVALUATION OF A CRYPTANALYTIC ATTACK

6.1 HERLESTAM'S METHOD

Tore Herlestam of Sweden's Department of Signal Security proposed [24] a method which would permit one to recover a plaintext from a ciphertext produced using the RSA public-key cryptosystem without knowledge of the decryption exponent d or the prime factors of n, i.e., P and Q . Herlestam's method consists of finding two polynomials in e, the public encryption exponent, namely Z(e), and X(e) such that:

$$Z(e)=e*X(e) \qquad (6.1)$$

To crack the cryptosystem one would try to find a polynomial Z(e) so that for a given ciphertext C the

following equation holds.

$$C \equiv C^{Z(e)} \pmod{n} \tag{6.2}$$

If successful then the plaintext M may be recovered from

$$M \equiv C^{X(e)} \pmod{n}. \tag{6.3}$$

If the RSA encryption function is specified by F_e and the corresponding decryption function F_d as F_e^{-1} it is easy to construct $F_{Z(e)}$ when Z(e) is a polynomial since $F_{e'+e''}(M) = F_{e'}(M) F_{e''}(M) \pmod{n}$, in particular $F_{e^r} = F_e^r$. If a C happens to satisfy equation 6.3 it is a fixed-point of $F_{Z(e)}$. The usefulness of the method depends heavily on the number of fixed-points for $F_{Z(e)}$. The number of fixed-points $I_{Z(e)}(n)$ for $F_{Z(e)}$ is

$$I_{Z(e)}(n) = I_{Z(e)}(P) I_{Z(e)}(Q) \tag{6.4}$$

where

$$I_{Z(e)}(P) = 1 + \gcd(Z(e)-1, P-1) \tag{6.5}$$

In order for the fixed-points to constitute a significant portion of all n possible texts, Z(e)-1 should have large common divisors with both P-1 and Q-1.

For the simple case when $Z(e)=e^r$ the number of fixed points will be at the maximum when $P-1=2*P'$, P' a prime and $Q-1=2*Q'$, Q' a prime and both P' and Q' divide $Z(e)-1$, i.e., $e^r \equiv 1 \pmod{P'}$ and $e^r \equiv 1 \pmod{Q'}$. If, in addition, $P'=2*P''+1$, P'' a prime and similarly for Q', (P'' and Q'' divide r) the number of fixed points will only be significant for prohibitively large r when Z(e) is chosen to be e^r. Herlestam discounts a proposal by Rivest [25] to avoid common factors in e^r-1 and P-1, claiming that it would only work for "unrealisticly small" r. Consider though that r need not be too large before e^r really blows up into a very very large number. Herlestam nonetheless considers performing the calculation of equation 6.2 for an $r=10^6$ as not impractical. In any case Herlestam points out that factors in e^r-1 of the form e^w-1, where w divides r are not considered explicitly.

Herlestam recommends that even when P is chosen such that $P=2*P'+1$, P' a prime, $P'=2*P''+1$, P'' a prime (and, similarly, for Q) then the cryptosystem can be cracked using a Z(e) of the form e^a+e^b where a=1,2,.....,m, and b=1,2,.....k, m and k being small. This method is claimed to be able to recover about 25% of all plaintexts. The odds can be roughly doubled by complementing. Since e is odd, $(n-M)^e \equiv n-C \pmod{n}$ Therefore, equation 6.2 may be

rewritten as

$$n-C \equiv (n-C)^{Z(e)} \pmod{n} \qquad (6.6)$$

so that

$$M \equiv n-(n-C)^{X(e)} \pmod{n}. \qquad (6.7)$$

6.2 PROGRAM DECRYP

In order to test Herlestam's attack the FORTRAN program DECRYP was written to take a test plaintext file, encrypt it, and then for each record of ciphertext calculate $C^{Z(e)} \pmod{n}$, where $Z(e)= e^{a}+e^{b}$. If for some small value of a and b $C^{Z(e)} \pmod{n} \equiv C$ then the ciphertext record in question could be deciphered by calculating the result of $C^{X(e)} \pmod{n}$, where $X(e)= e^{a-1}+e^{b-1}$. Herlestam's method would be considered successful if DECRYP succeeded in finding polynomials Z(e) using small a and b so that equation 6.2 holds for at lest 25% of the records of the enciphered test file. When DECRYP finds a Z(e) satisfying equation 6.2 it proceeds to the next record even if a or b have not reached their permitted maximum. This is done to avoid wasting time looking for additional solutions. DECRYP accepts a user named plaintext file, encrypts it based on the number of characters per record

specified by the user, and then proceeds to try to find a Z(e) satisfying equation 6.2. If successful DECRYP prints the value of a and b.

Since e^a+e^b equals e^b+e^a, DECRYP varies a and b such that as a=1,2,3,.....m, b=1,.....a. This prevents unnecessary, and expensive repitition of calculations. A tremendous amount of time is also saved by noting that:

$$C^{e^a+e^b} \pmod n \equiv C^{e^a} * C^{e^b} \pmod n \equiv (C^{e^a} \pmod n) * (C^{e^b} \pmod n) \pmod n$$

This indicates that one need only perform an exponentiation and reduction modulo n each time a is incremented if the previous (lesser values) of $C^{e^a} \pmod n$ have been saved. For example, when a is increased to 4 say, it is then necessary to calculate 6.2 for Z(e)= e^4+e^4, e^4+e^3, e^4+e^2, and e^4+e^1, but this can easily be done by multiplying the result of $C^{e^4} \pmod n$ by itself and the previously stored values of $C^{e^a} \pmod n$ when a=1,2, and 3. DECRYP stores these previously calculated values in a random access file one value per record. The time associated with reading a record and performing the multiplication is significantly less than repeating the exponentiation and reduction modulo n. Nevertheless

DECRYP still requires a lot of CPU time to execute. The values of a and b were limited to < 36, and the test plaintext file was limited to 240 characters. This file which appears in Table 6.1 was reblocked to form three files: one of 120 records with 2 characters per record, one of 60 records with 4 characters per record, and one of 40 records with 6 characters per record. It was designed so that in each of these forms there were no duplicate records to avoid repeating the same series of calculations. Beyond 6 characters per record the CPU time became excessive considering the experiments were being run in a time sharing environment.

```
RECORD                    PLAINTEXT
  1          @ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefg
  2          hijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./0
  3          123456789:;<=>??>=<;:9876543210/.-,+*)('
  4          &%$#"! ~}|{zyxwvutsrqponmlkjihgfedcba`_^
  5          ]\[ZYXWVUTSRQPONMLKJIHGFEDCBA@ZAYBXCWDVE
  6          UFTGSHRIQJPKOLNMAaBbCcDdEeFfGgHhIiJjKkLl
             ^                                      ^
        record start                          record stop
```

Table 6.1

Plaintext file used with program DECRYP

The results of the experiments using Herlestam's attack are given in table 6.2. These indicate that the method is not useful for breaking the RSA cryptosystem.

In only one case, i.e., n=59953 with e=53 was the attack successful in achieving near a 25% rate of cracking. For all other runs less than 2% of the plaintext records could be recovered.

These results should be considered in the light of the CPU time required, which, on average was 15 minutes for the 2 char./record tests, 75 minutes for the 4 char./record tests, and 2 hours for the 6 char./record tests. Note that the time is highly dependent on the size of e, the number of records tested, and of course the maximum value of a and b. In that much CPU time n could easily have been factored. In fact in a rebuttal, Rivest [25] claims that Herlestam's attack may be viewed as a highly inefficient method for factoring n. If this is so then succesfully decrypting one ciphertext would allow one to factor n and thereby decrypt all ciphertexts after determining d. Rivest concludes that for large n the probability of finding a Z(e) that works is of the same order as attempting to break the code by exhaustive search for d. Herlestam concluded that the probability of $order_n(C)=P'Q'$, where $P'Q'$ divides $A'P'B'Q'$ (A',B', small primes, and P',Q' large primes), is inversely proportional to the number of divisors of $A'P'B'Q'=(P-1)(Q-1)$. Rivest argues that the fraction of divisors of (P-1)(Q-1) is not

3/4, but should be calculated from $(P'+Q'-1)/P'Q'$ which for large n will be on the order of 10^{-90}. For equation 6.2 to hold Z(e) must be divisible by $order_n(C)$. This requires that P'Q' divide Z(e), and Rivest states the probability of this is on the order of $(P'Q')^{-1}$. For an $n = 10^{80}$ P' and Q' would probably be on the order of 10^{38} so that it is obvious that for an n chosen to withstand attempts at factoring this attack does not present a threat.

For: P=73, Q=151, n=11023, a,b<26, 2char./rec. 120 records

e	No. Rec. Decipherable	Values of (a,b)
29	none	
4469	none	
41	none	
3161	none	
61	none	
3541	none	
97	1	(2,1)
5233	1	(1,1)

For: P=167, Q=359, n=59953, a,b<26, 2char./rec. 120 rec.

e	No. Rec. Decipherable	Values of (a,b)
43	2	(20,8) (11,5)
29023	2	(12,10) (7,3)
53	28	(25,25) each rec.
7849	2	(19,4) (8,1)
61	2	(16,14) (12,3)
12665	2	(9,5) (8,6)
73	2	(13,9) (18,15)
9769	2	(7,2) (6,1)
89	2	(25,4) (11,10)
17361	2	(10,7) (4,1)

NOTE: For each of the e used the same two records were found to be decipherable.

Table 6.2

Results of experiments using program DECRYP.

For: n=547009091, a,b<31 4 char./rec. 60 records

e	No. Rec. Decipherable	Values of (a,b)
178956847	none	
82622431	none	

For: n=5386573295209, a,b<36, 6 char./rec. 40 records

e	No. Rec. Decipherable	Values of (a,b)
178956847	none	
1935462763183	none	

Table 6.2 cont.

Results of experiments using program DECRYP.

CHAPTER 7

CONCLUSIONS AND FUTURE AREAS OF STUDY

There is a tremendous future for the marriage of cryptology and computers. For the present conventional cryptology, in the form of the DES, provides adequate security for stored and transmitted data/information. In the future the DES must be replaced or strengthened. In the meantime the problem of key distribution should be analysed more rigorously. Game theory should be examined as a tool for evaluating different key distribution protocols.

The RSA public-key cryptosystem has been found to be secure against the cryptanalytic attack proposed by T. Herlestam. unfortunately the high level (FORTRAN) implementation of the system is too slow for use in a high volume electronic mail system. The key generation

programs are useful however. The RSA cryptosystem could be used to make an insecure channel secure for the rapid and inexpensive transmission of a conventional cryptosystem's keys, e.g., the DES. Herlestam [24] also suggests a method of breaking the trapdoor knapsack public-key cryptosystem. This should be investigated.

Research needs to be done on implementing the RSA cryptosystem on specially designed hardware. Still it is unlikely that encryption rates approaching the several million bits per second rates of the DES will ever be achieved.

While the RSA cryptosystem may never be useful for encrypting entire messages in a high volume electronic mail system, it may still be of value for implementing signatures. The problem of producing a short signature which uniquely identifies a plaintext message is an area for further research.

If a short (say 80-digit) signature could be quickly derived from a plaintext message that would uniquely (or at least, with high probability) identify the plaintext, then the RSA cryptosystem could be used in association with the DES to provide security, privacy, and authentication capability. The encryption of an 80-digit

signature by the RSA cryptosystem would only require about a second. Decryption would only be necessary if a dispute should arise.

For example, a possible method of creating a signature might be as follows. A sender would inform the mail system that he/she wished to sign a message. Suppose the message were:

GARY FISHER SIGNED THIS

The system would create an 80-digit signature (in general the message will be much longer than the signature) that uniquely identifies the above plaintext. For example, the system might put the date, time, sender's social security number, and receiver's social security number into the signature. This would require 6+6+9+9=30 digits. The remaining 50 digits would be a number derived from the plaintext. It is suggested that to do this the system convert the message to a number by some substitution known only to the system's operators (e.g., the government). The date, time, etc. could then be used as a seed to a secret (known only to the government) random number generator. The series of numbers generated would then be used to produce a (near) unique number from the numerical form of the plaintext. Suppose we convert the plaintext to a number by substituting 1 for A, 2 for B,....26 for Z.

Then assume the random number series produced were 3,8,4,1,5,4,6,.... Then one possible method of generating the signature number is illustrated in figure 7.1.

G A R Y F I S H E R S I G N E D T H I S

as a series of numbers is:

7,1,18,25,6,9,19,8,5,18,19,9,7,14,5,4,20,8,9,19

using the random numbers the signature algorithm produces:

(7+1)x18 + (25+6+9+19+8+5+18)x19 + (9+7+14)x5 +

(20+8+9+19)

=

(8x18) + (90x19) + (30x5) + (56)

=

146 + 1710 + 150 + 56 = 2062

Figure 7.1

Creating a signature number.

This algorithm works this way:

1. Given a series of random numbers $r_1 r_2 r_3$.....

2. Set k=1

3. For the next r_k-1 plaintext numbers form the sum then multiply it by the r_kth plaintext number to form a product. If r_k=1, consider the plaintext number to be multiplied by unity. If the r_k is larger than the number of plaintext characters just sum the last few plaintext numbers and consider them multiplied by unity.

4. If all plaintext numbers have been used go to step 5, otherwise k=k+1 and go to step 3.

5. Sum all the products formed in step 3 the result is the signature number. STOP.

The signature number is appended to the other information in the signature; any remaining spaces are padded with zeroes. The signature is then encrypted using the sender's secret RSA decryption key. Both a clear and enciphered form of the signature is then sent along with the associated plaintext; all being encrypted using the DES if security and privacy are also desired.

With the above method, should a dispute arise, a court could reproduce the random number stream from the seed in the clear form of the signature, convert the plaintext in question to a number using the government's secret substitution method and go through the process of generating the signature number. If the plaintext is different from the one the signature was developed for, then the number generated by the court will not be the same as the one in the signature (most likely). Note that the receiver could not forge a signature without knowing the senders secret decryption exponent d.

With a system like the one outlined above it would be very difficult to produce a forged plaintext (especially with the desired content) which matched the signature of the legitimate plaintext even if one knew the government's substitution scheme and/or the method of generating random numbers. Both the substitution scheme and the random number generator could change often, the date in the signature revealing which system was used. The above system also allows signature numbers for very long plaintexts to be developed before the 50-digit limit is exceeded. In fact there is room for additional information in the signature. One useful item might be the total number of characters in the plaintext.

Until a fast public-key cryptosystem is developed, the implementation of authentication capability using such systems while relying on conventional cryptography to provide security and privacy is an area that should be studied. The development of public-key cryptosystems is still in its infancy. NP-complete problems offer many opportunities to develop such systems.

APPENDIX I

FLOWCHARTS OF MULTIPRECISION ALGORITHMS

NOTES ON THE ALGORITHMS

1. All algorithms are for operations on two positive integer operands which are stored in the one dimensional (FORTRAN) arrays NONE and NTWO.

2. In the flowcharts all variables and arrays that begin with an I, J, K, L, M, or N should be considered to function in all mathematical expressions as integer variables and arrays function in FORTRAN expressions. All others act as FORTRAN real variables or arrays.

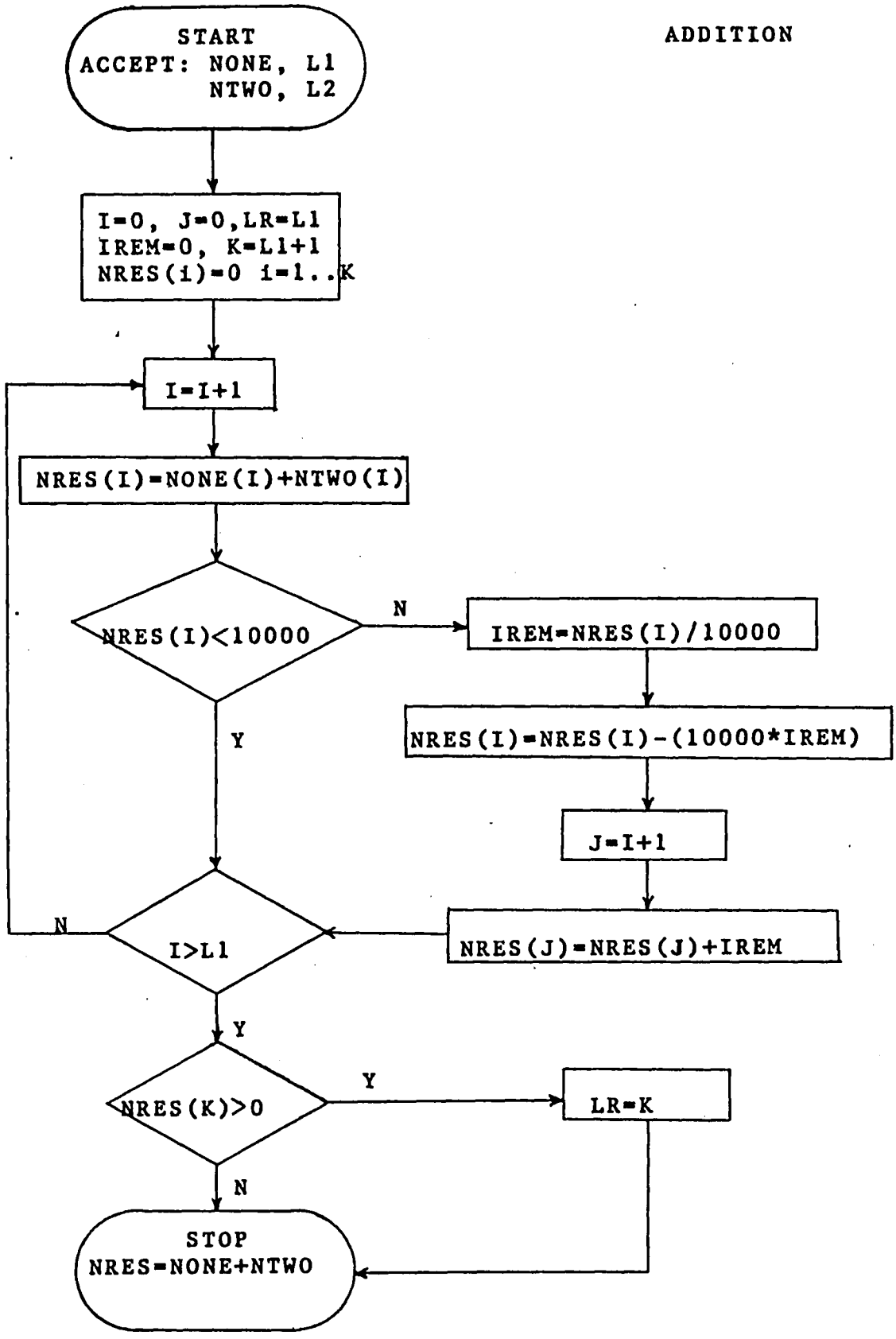
3. The result of each algorithm is a positive integer stored in the one dimensional FORTRAN array NRES. In the modulo and division algorithms NRES holds the remainder; the quotient in the division algorithm is stored in the array NQUO.

4. The number of array elements used are designated by the integer variables L1, L2, LR, and LQ for NONE, NTWO, NRES, and NQUO respectively. Low order digits are stored in the first array element and consecutive

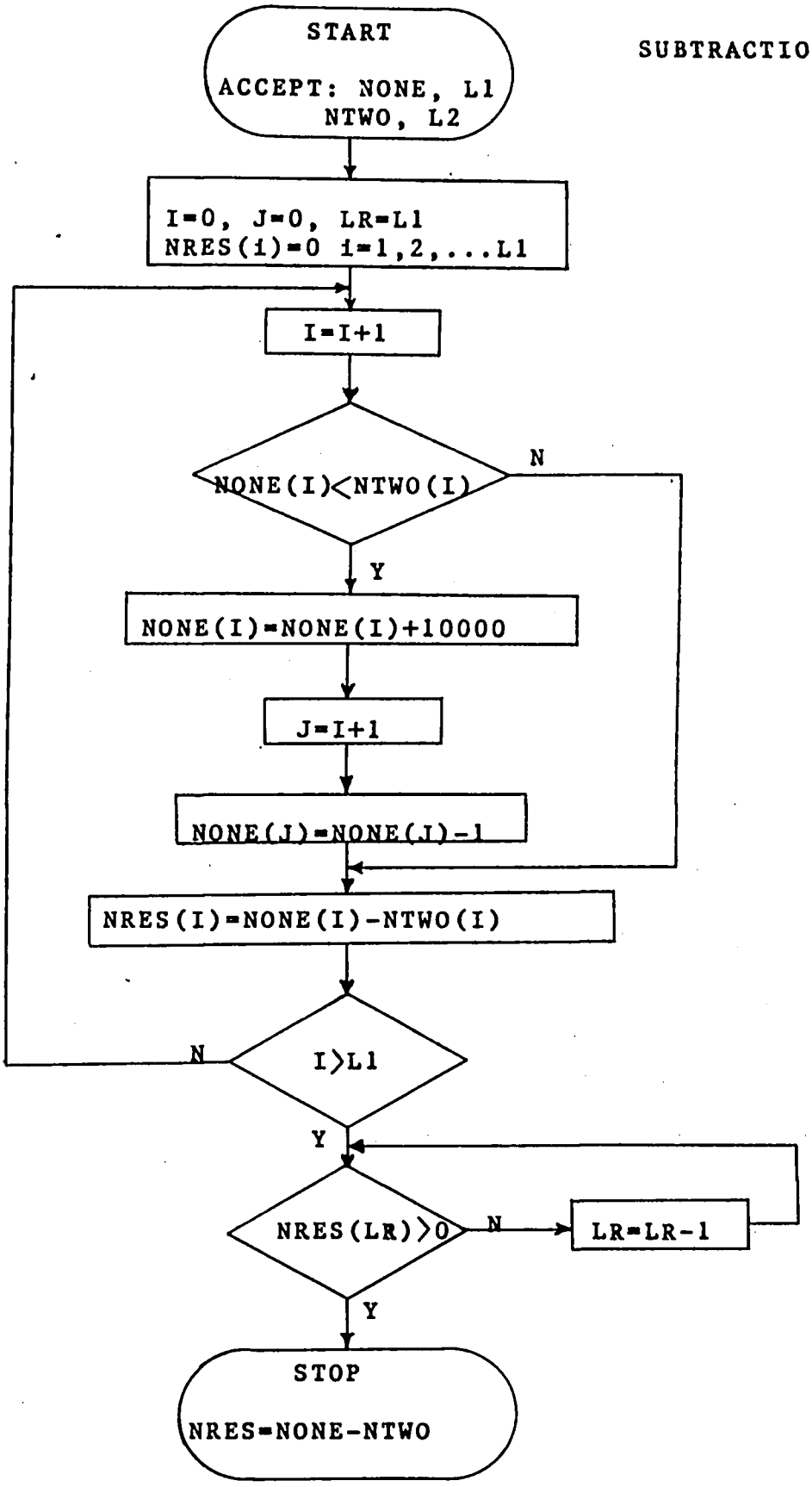
digits in consecutive elements, storing four digits per element. For example, if NONE=1,234,567,890 then NONE(1)=7890 NONE(2)=3456 NONE(3)=0012 and L1=3. Unused array elements are assumed equal to zero at the start of the algorithm.

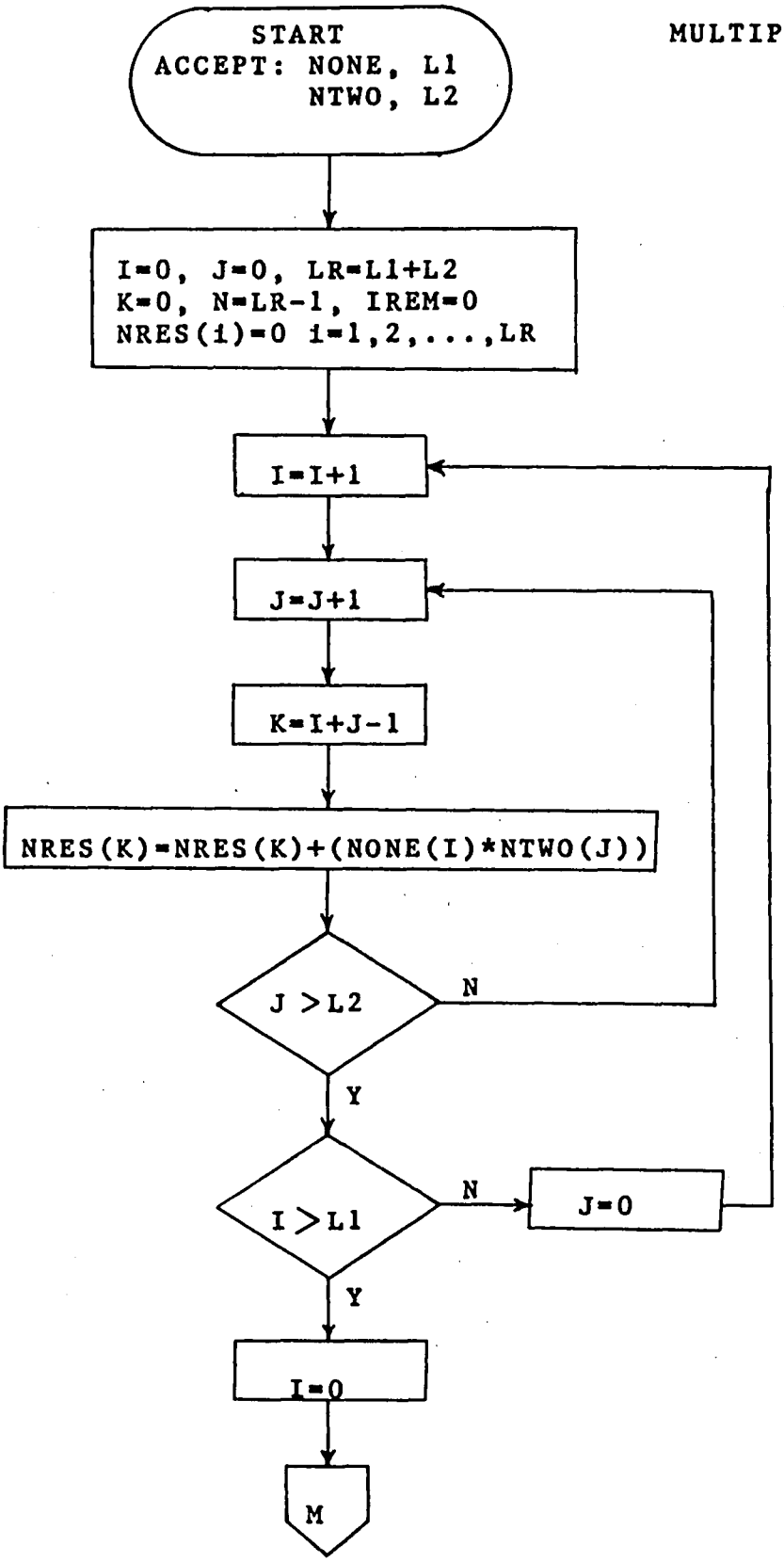
5. For the addition, subtraction, modulo, and division algorithms NONE≥NTWO in the flowcharts.

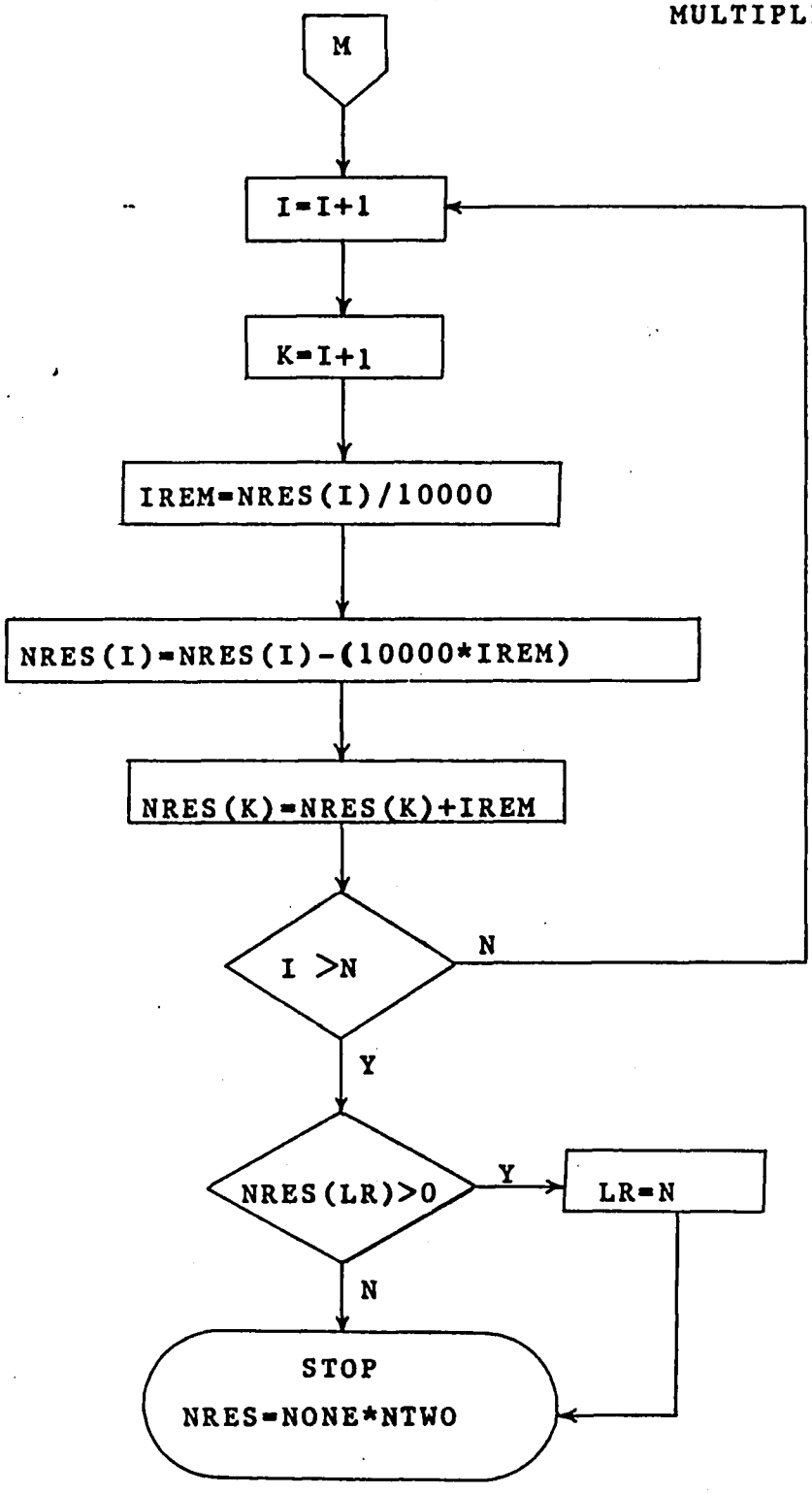
ADDITION


START
ACCEPT: NONE, L1
NTWO, L2
I=0, J=0,LR=L1
IREM=0, K=L1+1
NRES(i)=0 i=1..K
I=I+1
NRES(I)=NONE(I)+NTWO(I)
NRES(I)<10000
N
IREM=NRES(I)/10000
NRES(I)=NRES(I)-(10000*IREM)
J=I+1
Y
I>L1
N
NRES(J)=NRES(J)+IREM
Y
NRES(K)>0
Y
LR=K
N
STOP
NRES=NONE+NTWO

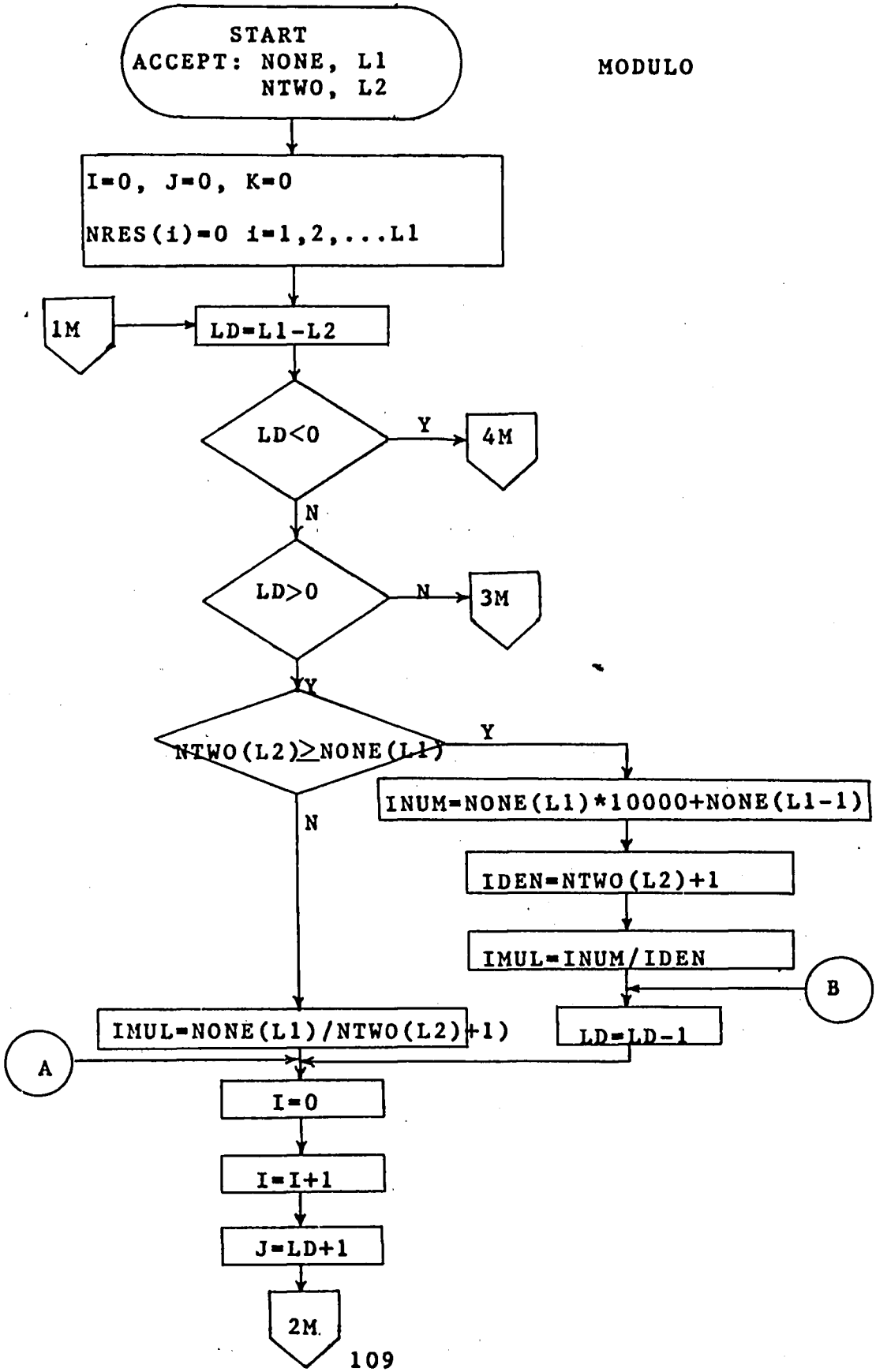
SUBTRACTION



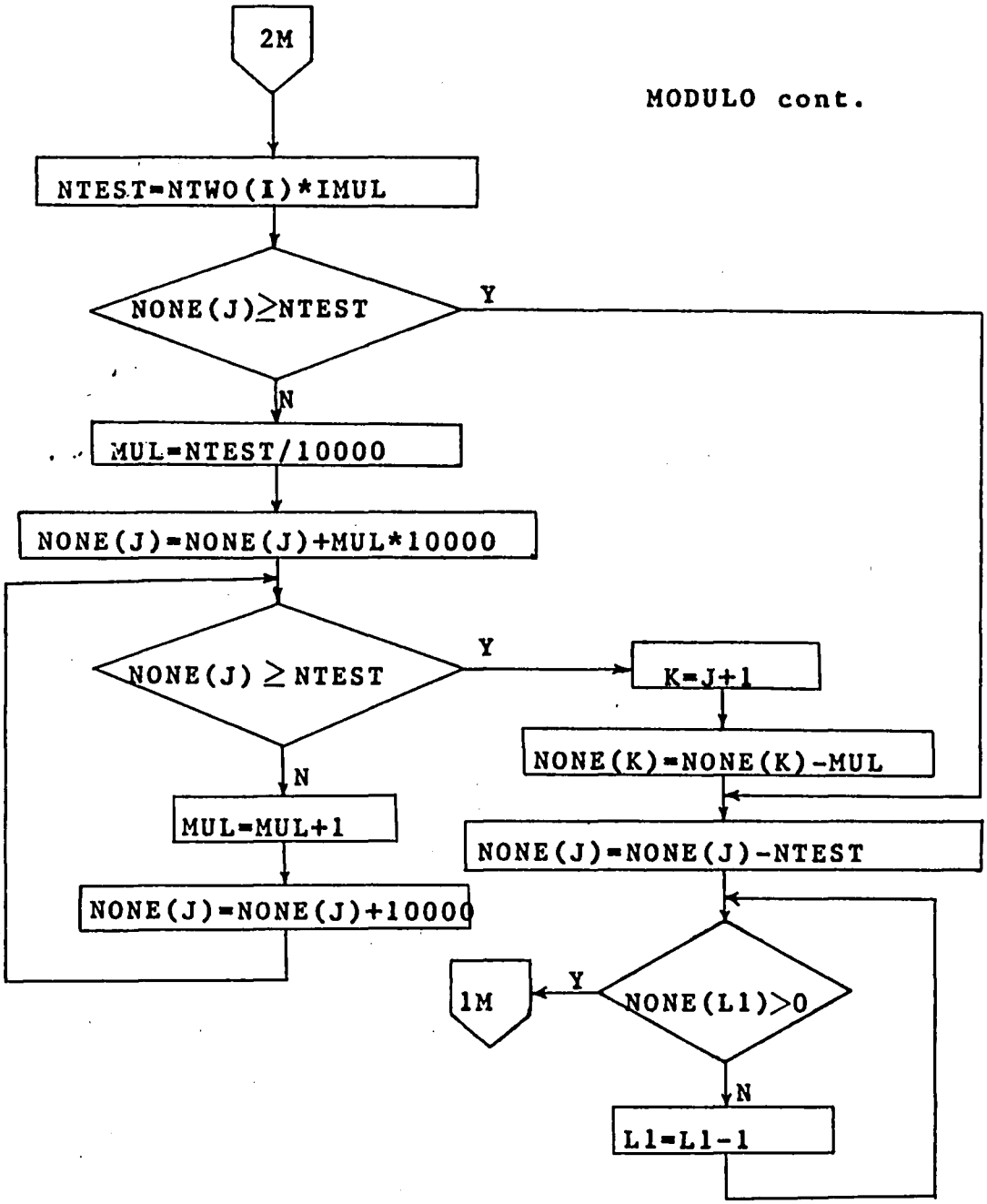

START
ACCEPT: NONE, L1
NTWO, L2
I=0, J=0, LR=L1+L2
K=0, N=LR-1, IREM=0
NRES(i)=0 i=1,2,...,LR
I=I+1
J=J+1
K=I+J-1
NRES(K)=NRES(K)+(NONE(I)*NTWO(J))
J >L2
N
Y
I >L1
N
J=0
Y
I=0
M

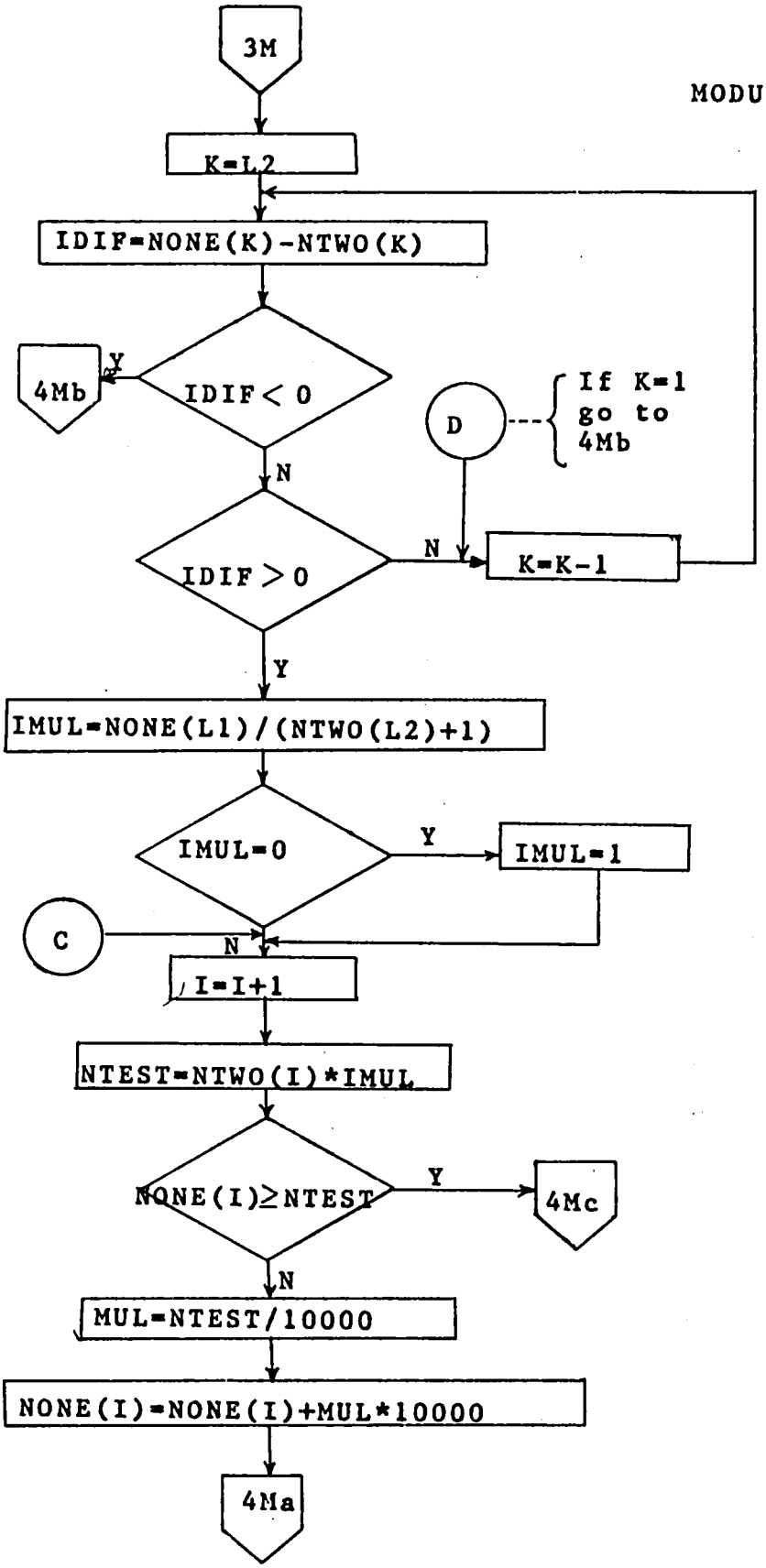

M
I=I+1
K=I+1
IREM=NRES(I)/10000
NRES(I)=NRES(I)-(10000*IREM)
NRES(K)=NRES(K)+IREM
I >N
N
Y
NRES(LR)>0
Y
LR=N
N
STOP
NRES=NONE*NTWO

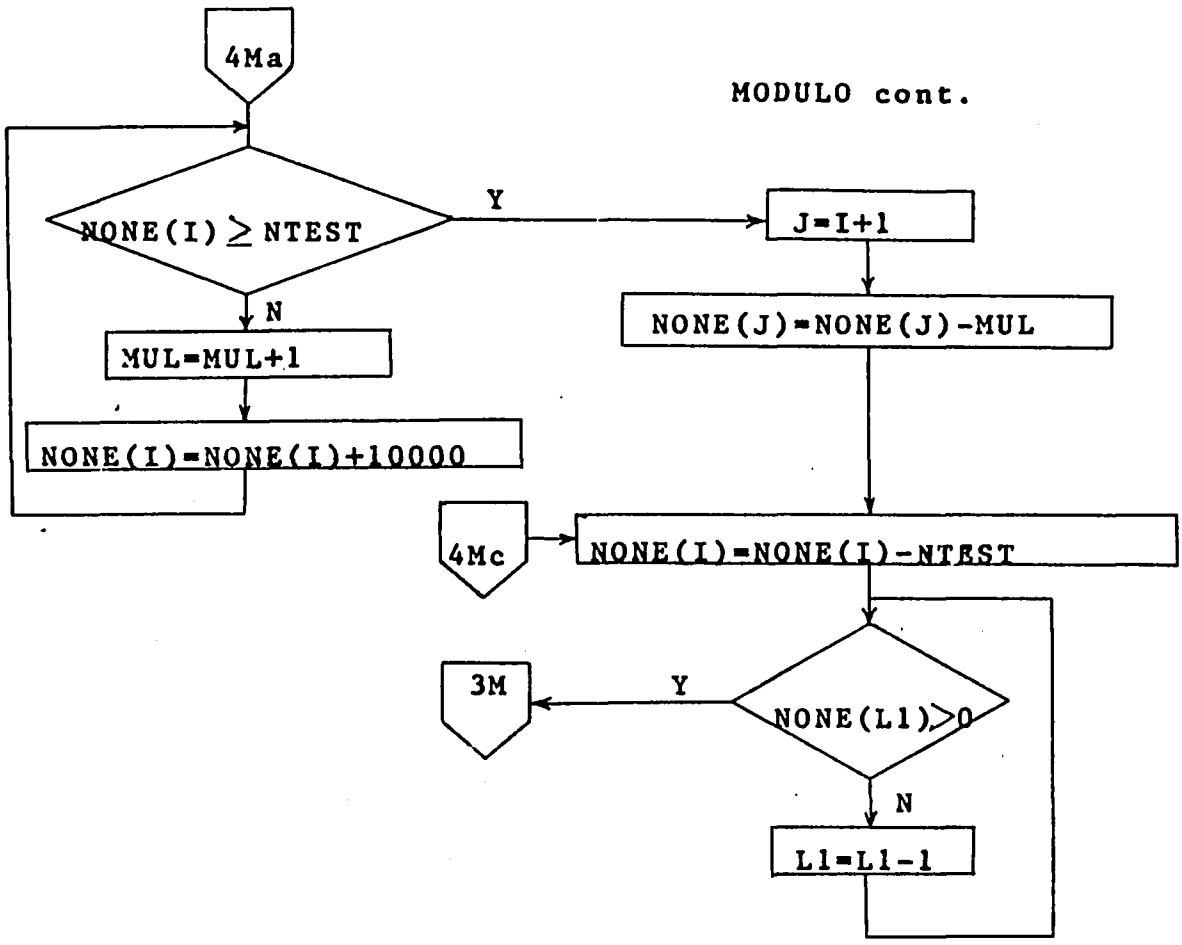
MODULO

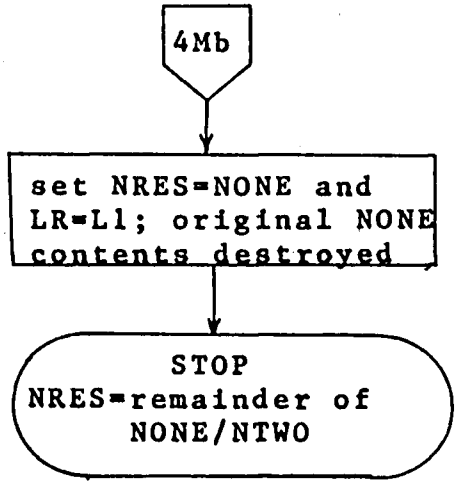

START
ACCEPT: NONE, L1
NTWO, L2
I=0, J=0, K=0
NRES(i)=0 i=1,2,...L1
1M
LD=L1-L2
LD<0
Y
4M
N
LD>0
N
3M
Y
NTWO(L2)≥NONE(L1)
Y
INUM=NONE(L1)*10000+NONE(L1-1)
IDEN=NTWO(L2)+1
IMUL=INUM/IDEN
B
LD=LD-1
N
IMUL=NONE(L1)/NTWO(L2)+1)
A
I=0
I=I+1
J=LD+1
2M

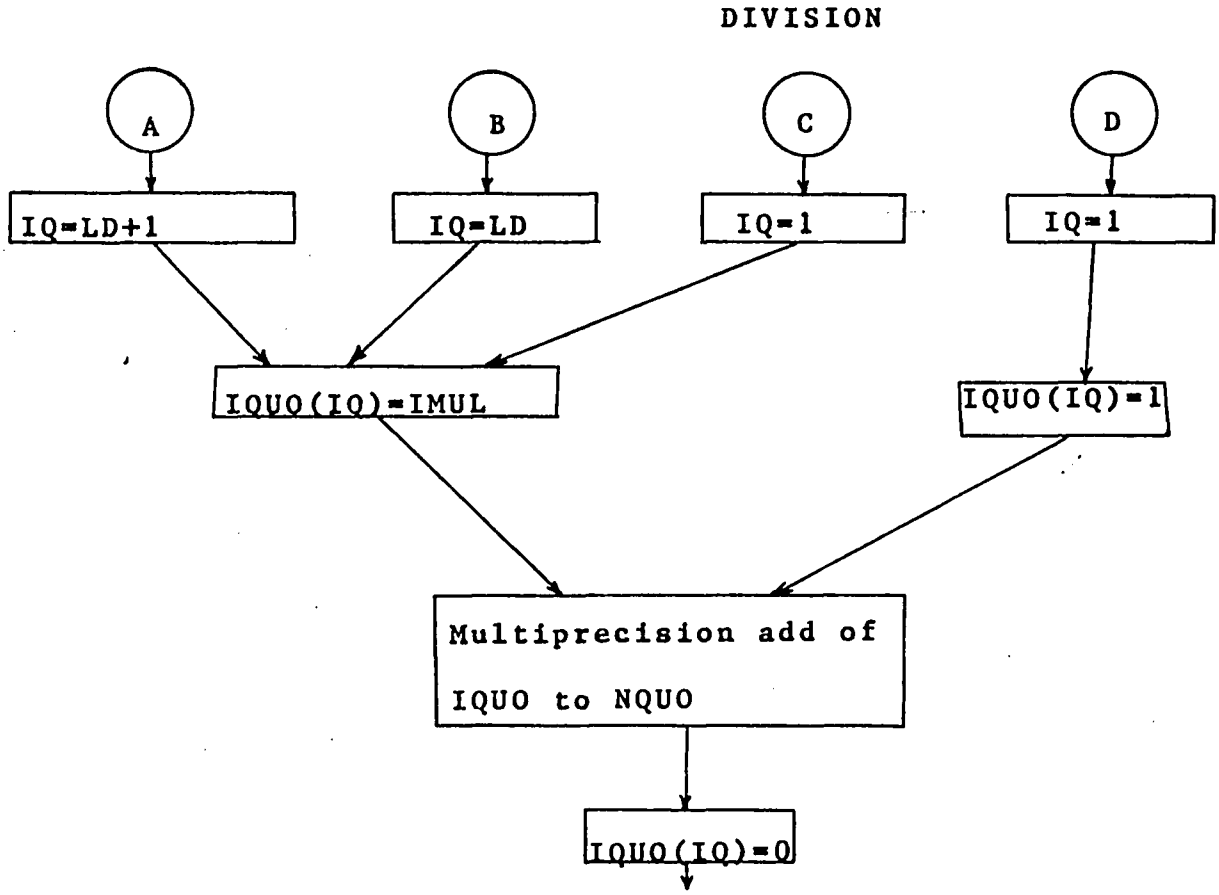
MODULO cont.


2M
NTEST=NTWO(I)*IMUL
NONE(J)≥NTEST
Y
N
MUL=NTEST/10000
NONE(J)=NONE(J)+MUL*10000
NONE(J) ≥ NTEST
Y
K=J+1
NONE(K)=NONE(K)-MUL
N
MUL=MUL+1
NONE(J)=NONE(J)-NTEST
NONE(J)=NONE(J)+10000
1M
Y
NONE(L1)>0
N
L1=L1-1


3M
K=L2
IDIF=NONE(K)-NTWO(K)
IDIF < 0
Y
4Mb
N
D
If K=1
go to
4Mb
IDIF > 0
N
K=K-1
Y
IMUL=NONE(L1)/(NTWO(L2)+1)
IMUL=0
Y
IMUL=1
C
N
I=I+1
NTEST=NTWO(I)*IMUL
NONE(I)≥NTEST
Y
4Mc
N
MUL=NTEST/10000
NONE(I)=NONE(I)+MUL*10000
4Ma


4Ma
MODULO cont.
NONE(I) ≥ NTEST
Y
J=I+1
N
NONE(J)=NONE(J)-MUL
MUL=MUL+1
NONE(I)=NONE(I)+10000
4Mc
NONE(I)=NONE(I)-NTEST
3M
Y
NONE(L1)>0
N
L1=L1-1


4Mb
set NRES=NONE and
LR=L1; original NONE
contents destroyed
STOP
NRES=remainder of
NONE/NTWO



For division define two additional arrays NQUO and IQUO and their length variables LQ and IQ, and insert the above logic in the modulo algorithm where shown. At the end of the algorithm NQUO will contain the quotient from the division of NONE by NTWO.

APPENDIX II

SAMPLE PROGRAM EXECUTIONS

NOTES

Underlined text is what was typed by the user. Text in parentheses is explanatory information, and does not appear on the user's terminal. A user typed carriage return is signified by [CR]. All programs were previously compiled.

PRIME

```
@EXECUTE PRIME[CR]
LINK:   Loading
[LNKXCT PRIME Execution]

THIS PROGRAM FINDS PRIME NUMBERS
STARTING AT AN INITIAL GUESS
HOW MANY DIGITS LONG WILL THE PRIME BE?
5[CR]

TYPE AN ODD NUMBER NOT ENDING IN A 5  5 DIGITS LONG
54549[CR]

 4549
 4551   (last 4 digits of each number tried)
 4553
 4557
THIS NUMBER IS PRIME
   54559
TO CONTINUE THE SEARCH TYPE C
[CR]    (a C typed will cause PRIME to continue executing)


STOP
```

NEXTPR

```
@EXECUTE NEXTPR[CR]
LINK:   Loading
[LNKXCT NEXTPR Execution]

THIS PROGRAM PRODUCES A PRIME NUMBER P SUCH THAT
P-1 HAS A PRIME FACTOR EQUAL TO A PRIME YOU SUPPLY
YOU MUST SUPPLY AN INITIAL PRIME NUMBER
HOW MANY DIGITS LONG IS IT?
5[CR]

TYPE A 'PRIME NUMBER   5 DIGITS LONG
54559[CR]

THIS NUMBER IS THE PRIME P
  982063
STOP
```

EFIND

```
@EXECUTE EFIND[CR]
LINK:   Loading
[LNKXCT EFIND Execution]

THIS PROGRAM FINDS A e WHICH IS RELATIVELY PRIME
TO THE TOTIENT FUNCTION OF TWO PRIMES P AND Q
YOU MUST SUPPLY P, Q AND AN INITIAL GUESS FOR e
TYPE THE LENGTH OF P
6[CR]

TYPE P
982063[CR]

TYPE THE LENGTH OF Q
3[CR]

TYPE Q
557[CR]

TYPE THE LENGTH OF YOUR GUESS NUMBER FOR e
9[CR]

TYPE AN ODD NUMBER e
178956847[CR]

THIS IS THE VALUE OF e
   178956847   (a lucky guess)
STOP
```

```
EXECUTE KEY[CR]
LINK:   Loading
[LNKXCT KEY Execution]

GIVEN TWO PRIME NUMBERS P AND Q AND A THIRD NUMBER
e RELATIVELY PRIME TO THE TOTIENT FUNCTION OF P
AND Q THIS PROGRAM COMPUTES A d AND n AND CREATES
ENCRYPTION AND DECRYPTION KEYFILES FOR PROGRAM RSA
TYPE THE NAME OF THE FILE TO CONTAIN THE DECRYPTION KEY
DKEY.DAT[CR]  (any acceptable DECsystem-20 file name)

TYPE THE NAME OF THE FILE TO CONTAIN THE ENCRYPTION KEY
EKEY.DAT[CR]  (any acceptable DECsystem-20 file name)

TYPE THE LENGTH OF P
6[CR]

TYPE P
982063[CR]

TYPE THE LENGTH OF Q
3[CR]

TYPE Q
557[CR]

TYPE THE LENGTH OF e
9[CR]

TYPE e
178956847[CR]

   546026472  (value of the totient function)
STOP  (if the d found is unacceptable there is an)
      (error message, try another e)


@TYPE EKEY.DAT[CR]  (to see the contents)
 3     178956847    (value of e)
 3     547009091    (value of n)


@TYPE DKEY.DAT[CR]
 2  82622431        (value of d)
 3     547009091    (value of n)
```

```
EXECUTE KEY[CR]
LINK:   Loading
[LNKXCT KEY Execution]

GIVEN TWO PRIME NUMBERS P AND Q AND A THIRD NUMBER
e RELATIVELY PRIME TO THE TOTIENT FUNCTION OF P
AND Q THIS PROGRAM COMPUTES A d AND n AND CREATES
ENCRYPTION AND DECRYPTION KEYFILES FOR PROGRAM RSA
TYPE THE NAME OF THE FILE TO CONTAIN THE DECRYPTION KEY
DKEY.DAT[CR]  (any acceptable DECsystem-20 file name)

TYPE THE NAME OF THE FILE TO CONTAIN THE ENCRYPTION KEY
EKEY.DAT[CR]  (any acceptable DECsystem-20 file name)

TYPE THE LENGTH OF P
6[CR]

TYPE P
982063[CR]

TYPE THE LENGTH OF Q
3[CR]

TYPE Q
557[CR]

TYPE THE LENGTH OF e
9[CR]

TYPE e
178956847[CR]

   546026472  (value of the totient function)
STOP  (if the d found is unacceptable there is an)
      (error message, try another e)


@TYPE EKEY.DAT[CR]  (to see the contents)
 3     178956847    (value of e)
 3     547009091    (value of n)


@TYPE DKEY.DAT[CR]
 2  82622431        (value of d)
 3     547009091    (value of n)
```

RSA (ENCRYPTION)

```
@EXECUTE RSA[CR]
LINK:   Loading
[LNKXCT RSA Execution]

FOR ENCRYPTION TYPE E FOR DECRYPTION TYPE D
E[CR]

TYPE SPECIFICATION OF FILE CONTAINING THE REQUIRED KEY
EKEY.DAT[CR]  (any acceptable DECsystem-20 file name)

TYPE SPECIFICATION OF FILE CONTAINING THE PLAINTEXT
M.DAT[CR]     (any acceptable DECsystem-20 file name)

CHARACTERS/RECORD OF FILE TO BE ENCRYPTED
NOTE: MUST BE AN EVEN INTEGER <  5     (based on n)
4[CR]

TYPE SPECIFICATION OF FILE TO CONTAIN THE ENCRYPTED TEXT
C.DAT[CR]     (any acceptable DECsystem-20 file name)

STOP
```

RSA (DECRYPTION)

```
@EXECUTE RSA[CR]
LINK    Loading
[LNKXCT RSA Execution]

FOR ENCRYPTION TYPE E FOR DECRYPTION TYPE D
D[CR]

TYPE SPECIFICATION OF FILE CONTAINING THE REQUIRED KEY
DKEY.DAT[CR]

TYPE SPECIFICATION OF FILE CONTAINING THE ENCRYPTED TEXT
C.DAT[CR]

TYPE SPECIFICATION OF FILE TO RECEIVE DECRYPTED MESSAGE
MDEC.DAT[CR]

STOP
(MDEC.DAT now equals M.DAT)
```

BIBLIOGRAPHY

1. Milliken, Donald D. Elementary Cryptography and Cryptanalysis, 2nd ed., New York University Bookstore, New york, 1943, p v.

2. Smith, Laurence D., "The History of Secret Writing", Cryptography The Science of Secret Writing, W W Norton & Company, Inc., New York, 1960, p 16.

3. Kahn, David The Code Breakers The Story of Secret Writing, The MacMillian Company, New York, 1970.

4. Costas, John P. "Cryptography in the Field." BYTE Vol. 4 , No. 3 , March 1979, pp. 56-63.

5. d'Agapeyeff, Alexander, "Commercial Codes", Codes And Ciphers, Oxford University Press, London, 1960, pp. 70-77.

6. Diffie, Whitfield, and Hellman, Martin E., "New Directions in Cryptography." IEEE Transactions on Information Theory, Vol. IT-22, No. 6, Nov. 1976, pp. 644-654.

7. Shannon, C. E. "Communication theory of secrecy systems." Bell System Tech. J., Vol. 28, Oct. 1949, pp. 656-715.

8. Wilkes, M. V., "Operation and Managerial Aspects of Time Sharing", Time-Sharing Computer Systems, American Elsevier, New york, 1972, pp. 91-93

9. Ehrsam, W. F., et. al., "A Cryptographic Key Management Scheme for implementing the Data Encryption Standard." IBM Systems Journal, Vol. 17, No. 2, 1978, pp. 106-125.

10. Diffie, Whitfield, and Hellman, Martin E., "Exhaustive Cryptanalysis of the NBS Data Encryption Standard." Computer, June 1977, pp. 74-84.

11. Kinnucan, Paul, "Data Encryption Gurus: Tuchman and Meyer." Mini-Micro Systems, Vol. 2, No. 9, Oct. 1978, pp. 56-57.

12. Matyas, S. M. et. al., "Generation, Distribution, and Installation of Cryptographic Keys." IBM Systems Journal, Vol. 17, No. 2, 1978, pp. 126-137.

13. Gladney, H. M., "Administrative Control of Computing Service." IBM Systems Journal, Vol. 17, No. 2, 1978, pp. 151-178.

14. Meyer, C. H. et. al., "Putting Data Encryption to Work." Mini-Micro Systems, Part 1 in Vol. 2, No. 9, Oct. 1978, pp. 46-52; Part 2 in Vol. 2, No. 10, December 1978, pp. 84-88.

15. Evans, Arthur, et. al., "A User Authentication Scheme Not Requiring Secrecy in the Computer." Communications of the ACM, Vol. 17, No. 8, August 1974, pp. 437-442.

16. Purdy, George B., "A High Security Log-in Procedure." Communications of the ACM, Vol. 17, No. 8, August 1974, pp. 442-445.

17. Merkle, R. C. and Hellman, M. E., "Hiding Information and Signatures in Trapdoor Knapsacks." IEEE Transactions on Information Theory, Vol. IT-24, No. 5, September 1978, pp. 525-530.

18. Rivest, R. L. et. al., A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, MIT Laboratory for Computer Science Technical Memo LCS/TM82, 1977.

19. Knuth, Donald E., The Art of Computer Programming, Vol 2 Seminumerical Algorithms, Addison-Wesley Publishing Company,Reading Mass., 1969.

20. Solovay, R. and Strassen, V. "A Fast Monte-Carlo Test for Primality", SIAM Journal on Computing, Vol. 6, No. 1, March 1977, pp. 84-85.

21. Solovay, R. and Strassen, V. "Erratum: A Fast Monte-Carlo Test for Primality", SIAM Journal on Computing, Vol. 7, No. 1, February 1978, pg. 118.

22. Lehmer, D. N., "On the Converse of Fermat's Theorem", American Mathematical Monthly, Vol. 43, 1936, pp. 347-349.

23. Lehmer, D. N., List of Prime Numbers From 1 to 10,006,721, Hafner Publishing Co., New York. 1956.

24. Herlestam, Tore, "Critical Remarks on Some Public-Key Cryptosystems", BIT, Vol. 18, 1978, pp. 493-496.

25. Rivest, R. L., "Remarks on a Proposed Cryptanalytic Attack on the M.I.T. Public-Key Cryptosystem", Cryptologia, Vol. 2, No. 1, 1978, pp. 62-65.

26. Rivest, R. L., "Critical Remarks on "Critical Remarks on Some Public-Key Cryptosystems" by T. Herlestam", BIT, Vol. 19, 1979, pp. 274-275.

VITA

NAME: Gary Carter Fisher
DATE OF BIRTH: November 25, 1956
PLACE OF BIRTH: Hackensack, New Jersey

PARENTS:
FATHER: Joseph M. Fisher
MOTHER: Yvonne B. Fisher

EDUCATION:
HIGH SCHOOL: Cherry Hill High School East
LOCATION: Cherry Hill, New Jersey
FROM: August 1970
TO: June 1974

UNIVERSITY (UNDERGRADUATE): Lehigh University
LOCATION: Bethlehem, Pennsylvania
FROM: August 1974
TO: May 1978
DEGREE: BS Industrial Engineering

UNIVERSITY (GRADUATE): Lehigh University
LOCATION: Bethlehem, Pennsylvania
FROM: August 1978
TO: January 1980
DEGREE: MS Industrial Engineering

Mr. Fisher has been a teaching assistant in the Industrial Engineering Department at Lehigh University where he taught the laboratory for a course in work systems and an introductory course in manufacturing engineering.

He presently lives in Rutherford, New Jersey and is seeking a career as an operations researcher or systems analyst.